DYNIX/ptx Reference Manual
Section 3

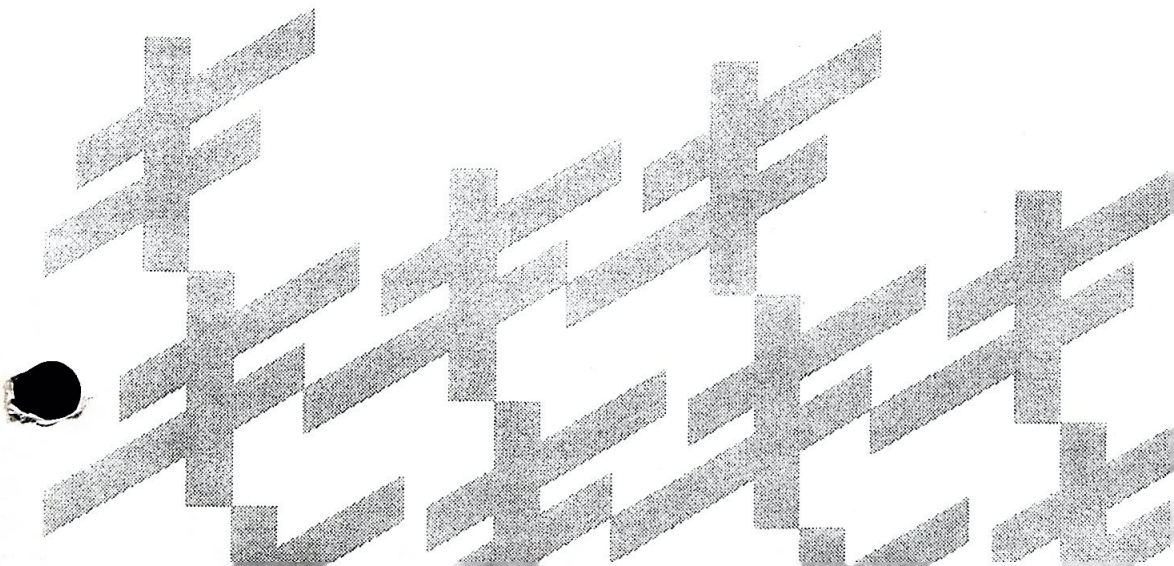
1003-49233-01

THIS SHEET LIFTER
MUST BE PLACED ON
TOP OF CONTENTS

GRAPHIC PRODUCTS
Chicago, Illinois 60609

DYNIX/ptx®
Reference Manual
Section 3

1003-49233-01
Sequent Computer Systems, Inc.



ACT is a trademark of Micro-Term.
Ann Arbor is a trademark of Ann Arbor Terminals.
Beehive is a trademark of Beehive International.
Concept is a trademark of Human Designed Systems.
Dataphone is a registered trademark of AT&T.
DEC is a trademark of Digital Equipment.
Diablo is a registered trademark of Xerox.
Digital is a trademark of Digital Equipment.
DOCUMENTER'S WORKBENCH is a trademark of AT&T.
DYNIX and DYNIX/ptx are registered trademarks of Sequent Computer Systems, Inc.
HP is a trademark of Hewlett-Packard.
i386 is a trademark of Intel Corp.
LSI is a trademark of Lear Siegler.
Micro-term and MIME are trademarks of Micro-Term.
MULTIBUS is a trademark of Intel Corp.
PDP is a trademark of Digital Equipment.
Sequent is a trademark of Sequent Computer Systems, Inc.
Symmetry is a registered trademark of Sequent Computer Systems, Inc.
Tektronix and Tektronix 4010 are registered trademarks of Tektronix.
Teleray is a trademark of Research.
Teletype is a registered trademark of AT&T.
TeleVideo is a registered trademark of TeleVideo Systems.
UNIX is a registered trademark of AT&T.
VAX is a trademark of Digital Equipment.
Versatec is a registered trademark of Versatec.
VT100 is a trademark of Digital Equipment.

Copyright © 1989, 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

About This Manual

Overview

The *DYNIX/ptx Reference Manual* is for programmers working in the DYNIX/ptx operating system environment. This manual describes the programming features of the operating system and assumes that the audience is familiar with the operating system. It does not provide a general overview of the operating system or implementation details of the system.

Organization

The *DYNIX/ptx Reference Manual* is contained in five binders organized as follows:

- Section 1 A-F **Commands**
- Section 1 G-Z **Commands**
- Section 2 **System Calls**
  Section 2SEQ **Sequent System Calls**
  Section 3 **Subroutines**
  - Section 3C **C Library Routines**
  - Section 3M **Mathematical Library Routines**
  - Section 3N **Networking Support Utilities**
  - Section 3PPS **Sequent Parallel Programming Library**
  - Section 3S **Standard I/O Library Routines**
  - Section 3SEQ **Sequent I/O Library Routines**
  - Section 3X **Special Routines**
- Section 4 **File Formats**
  Section 5 **Miscellaneous Facilities**
- Section 7 **Special Files**
  Section 8 **Stand-alone Utilities**

Each binder also contains the following items:

- **Table of Contents** that alphabetically lists the contents of the entire manual set
- **Permuted Index** that lists each programming feature in the entire manual set

The permuted index is a list of keywords, given alphabetically in the second of three columns, together with the context in which each keyword is found. The following shows four separate entries illustrating four ways that the **cat** command can be looked up in the permuted index. Note that you can find the **cat** command by looking up "cat," "concatenate," "files," or "print."

```
                    cat: concatenate and print files.....cat(1)
                cat:  concatenate and print files..... cat(1)
concatenate and print  files.....cat(1)
     concatenate and  print files..... cat(1)
```

Keywords are either topical keywords or the names of manual entries themselves. Entries are identified with their section numbers shown in parentheses. The numbers are important because there is duplication of names among the sections. The right column lists the name of the manual page on which each keyword may be found. The left column contains useful information about the keyword.

The following paragraphs briefly describe the other sections of the manual.

Section 1: Commands describes operating system commands and commands that support C and other programming languages.

Section 2: System Calls describes the access to the services provided by the operating system kernel, including the C language interface.

Section 2SEQ: Sequent System Calls describes the access to the Sequent services provided by the operating system kernel, including the C language interface.

Section 3: Subroutines describes the available subroutines. Their binary versions reside in various system libraries in the directories */lib* and */usr/lib*. The introduction to Section 3 describes these libraries and the files in which they are stored.

Section 4: File Formats documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language structures corresponding to these formats can be found in the directories */usr/include* and */usr/include/sys*.

Section 5: Miscellaneous Facilities contains a variety of descriptions including descriptions of character sets, macro packages, and so forth.

Section 7: Special Files describes various special files that refer to specific hardware peripherals and operating system device drivers.

Section 8: Stand-alone Utilities describes stand-alone procedures.

References to manual pages with numbers other than those above mean that the utility is contained in the appropriate section of another manual. Each section consists of independent entries which are alphabetized, with the exception of the introductions that begin some sections. Section 3 is divided into subsections in alphabetical order by suffix. Some entries may describe several routines, commands, and so forth. In such cases, the entry appears only once, alphabetized under its *primary* name, the name that appears at the upper corners of each manual page.

Each manual page can be viewed online with the **man**(1) command. For example, enter **man shell** to view the **shell** manual page. Enter **man man** for more information about the **man** command.

Notational Conventions

Manual pages are based on a common format. The main headings are described here—not all of them appear in all cases:

- **NAME** gives the name(s) of the entry and briefly states its purpose. This entry is used to create the Permuted Index and the Table of Contents.
- **SYNOPSIS** summarizes the usage of the program being described. Conventions vary in some sections but are generally as follows:

- **Boldface** strings are literals and are to be typed just as they appear.
- *Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.
- Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as *name* or *file*, it refers to a *filename*.
- Ellipses ... are used to show that the previous argument prototype may be repeated.
- A final convention is used by the commands themselves. An argument beginning with a minus –, plus +, or equal sign = is often taken to be some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with –, +, or =.

- **CONFIGURATION ENTRY** shows how a driver is called out in a system configuration file.
- **DESCRIPTION** describes the entry, including its purpose and how it is used.
- **FIELDS** describe system file or record entries.
- **EXAMPLES** and **BINARY CONFIGURATION** give examples of usage.
- **FILES** list files that are part of a program, created by a program, or related to a program.
- **SEE ALSO** points to related information in other manual pages.
- **DIAGNOSTICS** discuss the diagnostic (error) messages that may be produced. Messages that are self-explanatory are not listed.
- **WARNINGS** contains warnings against improper user actions.
- **NOTES** give hints about the use of the utility.
- **BUGS** describe known bugs in the software.

Contents

1 Commands

intro(1) ..... introduction to maintenance commands and application programs
300(1): 300, 300s ..... handle special functions of DASI 300 and 300s terminals
4014(1) ..... paginator for the Tektronix 4014 terminal
450(1) ..... handle special functions of the DASI 450 terminal
accept(1M): accept, reject ..... allow or prevent LP requests
acct(1M): acctdisk, acctdusg, accton, acctwtmp ..... overview of accounting and miscellaneous accounting commands
acctcms(1M) ..... command summary from per-process accounting records
acctcom(1) ..... search and print process accounting files
acctcon(1M): acctcon1, acctcon2 ..... connect-time accounting
acctmerg(1M) ..... merge or add total accounting files
acctprc(1M): acctprc1, acctprc2 ..... process accounting
acctsh(1M): chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct ..... shell procedures for accounting
addcmdopt(1M) ..... add a command option to a menu
addformopt(1M) ..... add a form/SELECT window option to a menu
addmenuopt(1M) ..... add a menu option to a menu
admin(1) ..... create and administer SCCS files
apropos(1) ..... locate commands by keyword lookup
ar(1) ..... see *C and Language Tools Binder*
as(1) ..... see *C and Language Tools Binder*
asa(1) ..... interpret ASA carriage control characters
assist(1) ..... assistance using operating system commands
astgen(1) ..... program for generating/modifying ASSIST menus or command forms
at(1): at, batch ..... execute commands at a later time
awk(1) ..... pattern scanning and processing language
banner(1) ..... make posters
basename(1): basename, dirname ..... deliver portions of pathnames
bc(1) ..... arbitrary-precision arithmetic language
bcheckrc(1M) ..... system initialization procedures
bdflush(1M) ..... update daemon
bdiff(1) ..... big diff
bfs(1) ..... big file scanner
bomverify(1M) ..... perform an installation consistency check based on the BOM
bootflags(1M) ..... display/change system boot flags
bp(1M) ..... binary patcher
cal(1) ..... print calendar

calendar(1) ..... reminder service
captoinfo(1M) ..... convert a termcap description into a terminfo description
cat(1) ..... concatenate and print files
cb(1) ..... see *C and Language Tools Binder*
cc(1) ..... see *C and Language Tools Binder*
cd(1) ..... change working directory
cdc(1) ..... change the delta commentary of an SCCS delta
cflow(1) ..... see *C and Language Tools Binder*
checkeq(1) ..... check documents for correct eqn syntax.
checkfsys(1M) ..... check a filesystem
chmod(1) ..... change mode
chown(1): chown, chgrp ..... change owner or group
chroot(1M) ..... change root directory for a command
chrtbl(1M) ..... generate character classification and conversion tables
ci(1) ..... check in RCS revisions
ckbupscd(1M) ..... check filesystem backup schedule
clri(1M) ..... clear inode
cmp(1) ..... compare two files
co(1) ..... check out RCS revisions
col(1) ..... filter reverse line-feeds
comb(1) ..... combine SCCS deltas
comm(1) ..... select or reject lines common to two sorted files
compress(1): compress,
uncompress, zcat ..... compress and expand data
config(1M) ..... build system configuration files
coproc(1F): cocreate, cosend,
cocheck, coreceive, codestroy ..... communicate to a process
cp(1): cp, ln, mv ..... copy, link or move files
cpp(1) ..... see *C and Language Tools Binder*
cpio(1) ..... copy file archives in and out
cprs(1) ..... see *C and Language Tools Binder*
cpset(1) ..... copy file and set mode, user, and group IDs
crash(1M) ..... examine system images
cron(1M) ..... clock daemon
crontab(1) ..... user crontab file
crypt(1) ..... encode/decode
csh(1) ..... a shell (command interpreter) with C-like syntax
csplit(1) ..... context split
ct(1C) ..... spawn getty to a remote terminal
ctags(1) ..... create a tags file
ctrace(1) ..... see *C and Language Tools Binder*
cu(1C) ..... call another UNIX system
cut(1) ..... cut out selected fields of each line of a file
cxref(1) ..... see *C and Language Tools Binder*
date(1) ..... print and set the date
dbx(1) ..... see *C and Language Tools Binder*
dc(1) ..... desk calculator
dcheck(1M) ..... filesystem directory consistency check
dd(1M) ..... convert and copy a file
delta(1) ..... make a delta (change) to an SCCS file

deroff(1) ..... remove nroff, tbl, and eqn constructs
devbuild(1M): devbuild,
devdestroy ..... build and destroy virtual devices
devnm(1M) ..... device name
df(1M) ..... report number of free disk blocks and inodes
diff(1) ..... differential file comparison
diff3(1) ..... 3-way differential file comparison
dircmp(1) ..... directory comparison
dis(1) ..... see *C and Language Tools Binder*
diskusg(1M) ..... generate disk accounting data by user ID
dltsectbl(1M) ..... refresh security table
dmesg(1M) ..... collect system diagnostic messages to form error log
du(1M) ..... summarize disk usage
dump(1) ..... see *C and Language Tools Binder*
dump(1M) ..... incremental filesystem dump
dumpconf(1M) ..... dump kernel auto-configuration device table
dumpfs(1M) ..... dump filesystem information
ec(1M) ..... forms compiler
echo(1) ..... echo arguments
echo(1F) ..... put string on virtual output
ed(1): ed, red ..... text editor
edc(1M) ..... run an externally described command
edit(1) ..... text editor (variant of ex for casual users)
edquota(1M) ..... edit user quotas
efl(1) ..... Extended Fortran Language
egrep(1) ..... search a file for a pattern using full regular expressions
enable(1): enable, disable ..... enable/disable LP printers
env(1) ..... set environment for command execution
ex(1) ..... text editor
expr(1) ..... evaluate arguments as an expression
factor(1) ..... obtain the prime factors of a number
ff(1M) ..... list filenames and statistics for a filesystem
fgrep(1) ..... search a file for a character string
file(1) ..... determine file type
find(1) ..... find files
format(1M) ..... format disks or perform other defect management functions
fsck(1M) ..... filesystem consistency check and interactive repair
fsirand(1M) ..... install random inode generation numbers
fsplit(1) ..... split f77, ratfor, or efl files
fsstat(1M) ..... report filesystem status
fstyp(1M) ..... determine filesystem identifier
fuser(1M) ..... identify processes using a file or file structure
fwtmp(1M): fwtmp, wtmpfix ..... manipulate connect accounting records
gdev(1G): hpd, erase, hardcopy,
tekset, td ..... graphical device routines and filters
ged(1G) ..... graphical editor
gencc(1M) ..... create a front-end to the cc command
get(1) ..... get a version of an SCCS file
getfrm(1F) ..... returns the current frame number

getitems(1F) ..... returns a list of the currently marked menu items
getopt(1) ..... parse command options
getopts(1): getopts, getoptcvt ..... parse command options
getty(1M) ..... set terminal type, modes, speed, and line discipline
glossary(1) ..... definitions of common operating system terms and symbols
graph(1G) ..... draw a graph
graphics(1G) ..... access graphical and numerical commands
greek(1) ..... select terminal filter
grep(1) ..... search a file for a pattern
groups(1) ..... show group memberships
gutil(1G) ..... graphical utilities
help(1) ..... operating system Help Facility
helpadm(1M) ..... change the Help Facility database
hp(1) ..... handle special functions of Hewlett-Packard terminals
hpio(1) ..... Hewlett-Packard 2645A terminal tape file archiver
icheck(1M) ..... filesystem storage consistency check
id(1M) ..... print user and group IDs and names
ident(1) ..... identify files
indicator(1F) ..... display application specific alarms and/or the working indicator
infocmp(1M) ..... compare or print out terminfo descriptions
init(1M): init, telinit ..... process control initialization
install(1M) ..... install commands
installboot(1M) ..... installs a boot program on a disk
ipcrm(1) ..... remove a message queue, semaphore set, or shared memory ID
ipcs(1) ..... report inter-process communication facilities status
isismgr(1M) ..... ISIS file manager, updates text files in ISIS file format
join(1) ..... relational database operator
kill(1) ..... terminate a process
killall(1M) ..... kill all active processes
ksh(1): ksh, rksh ..... the KornShell, a standard/restricted command and programming language
labelit(1M) ..... provide labels for filesystems
ld(1) ..... see *C and Language Tools Binder*
lex(1) ..... generate programs for simple lexical tasks
line(1) ..... read one line
link(1M): link, unlink ..... link and unlink files and directories
lint(1) ..... see *C and Language Tools Binder*
list(1) ..... see *C and Language Tools Binder*
locate(1) ..... use keywords to identify an operating system command
login(1) ..... sign on
logname(1) ..... get login name
lorder(1) ..... see *C and Language Tools Binder*
lp(1): lp, cancel ..... send/cancel requests to the LP print service
lpadmin(1M) ..... configure the LP print service
lpfilter(1M) ..... administer filters used with the LP print service
lpforms(1M) ..... administer forms used with the LP print service

lpsched(1M): lpsched, lpshut,
lpmove ..... start/stop the LP print service and move requests
lpstat(1) ..... print information about the status of the LP print service
lpusers(1M) ..... set printing queue priorities
ls(1) ..... list contents of directory
m4(1) ..... macro processor
machid(1): i386, ns32000, pdp11,
u3b, u3b2, u3b5, vax ..... get processor type truth value
mail(1): mail, rmail ..... send mail to users or read mail
mailbug(1) ..... submit a bug report or enhancement request
mailx(1) ..... interactive message processing system
make(1) ..... maintain, update, and regenerate groups of programs
makealiases(1M): makealiases,
makewhatis ..... generate man page databases
makedev(1M) ..... make system special files
makekey(1) ..... generate encryption key
man(1) ..... display on-line man pages
mcs(1) ..... manipulate the object file comment section
menu(1M) ..... menu processor
menumsg(1M) ..... route user message to menu message queue
merge(1) ..... three-way file merge
mesg(1) ..... permit or deny messages
message(1F) ..... puts its arguments on message line
mkdir(1) ..... make directories
mkfs(1M) ..... construct a filesystem
mknod(1M) ..... build special file
mkpwdbm(1M) ..... make a database for fast password lookup
mkvtoc(1M) ..... populate VTOC on disks
monitor(1M) ..... monitor system activity
more(1) ..... view a file one full screen at a time
mount(1M): mount, umount ..... mount and unmount filesystems
mountall(1M): mountall,
umountall ..... mount, unmount multiple filesystems
mt(1) ..... magnetic tape manipulating program
mvdir(1M) ..... move a directory
nawk(1) ..... pattern scanning and processing language
ncheck(1M) ..... generate pathnames from i-numbers
newform(1) ..... change the format of a text file
newfs(1M) ..... construct a new file system
newgrp(1M) ..... log in to a new group
news(1) ..... print news items
nice(1) ..... run a command at low priority
nl(1) ..... line numbering filter
nm(1) ..... see *C and Language Tools Binder*
noage(1M): noage, nopff, noswap,
onproc ..... run a command with priority aging, PFF, swapping, or processor migration disabled
nohup(1) ..... run a command immune to hangups and quits
od(1) ..... octal dump
online(1M): online, offline ..... bring processors online and offline

pack(1): pack, pcat, unpack ..... compress and expand files
passmgmt(1M) ..... password files management
passwd(1) ..... change login password and password attributes
paste(1) ..... merge same lines of several files or subsequent lines of one file
pathconv(1F) ..... search Interpreter criteria for filename
pdbx(1) ..... see *C and Language Tools Binder*
pg(1) ..... file perusal filter for CRTs
pmap(1M) ..... display or alter parameters of virtual-to-physical mapping devices
pr(1) ..... print files
prof(1) ..... display profile data
prompt(1M) ..... prompt program
prs(1) ..... print an SCCS file
prtvtoc(1M) ..... print the VTOC of a block device
ps(1) ..... report process status
pwck(1M): pwck, grpck ..... password/group file checkers
pwconv(1M) ..... installs and updates /etc/shadow with information from /etc/passwd
pwd(1) ..... working directory name
pwdhd(1M) ..... daemon to oversee password database creation
pwfix(1M) ..... remove incorrectly formatted password entries
pwunconv(1M) ..... converts from a two to a one password file scheme
quota(1) ..... display disk usage and limits
quotacheck(1M) ..... filesystem quota consistency checker
quotaon(1M): quotaon, quotaoff ..... turn filesystem quotas on and off
ratfor(1) ..... rational Fortran dialect
rc0(1M) ..... run commands performed to stop the operating system
rc2(1M) ..... run commands performed for multi-user environment
rcs(1) ..... change RCS file attributes
rcsdiff(1) ..... compare RCS revisions
rcsintro(1) ..... introduction to RCS commands
rcsmerge(1) ..... merge RCS revisions
readfile(1F): readfile, longline ..... reads file and gets longest line
regcmp(1) ..... regular expression compile
regex(1F) ..... match patterns against a string
reinit(1F) ..... run an initialization file
repquota(1M) ..... summarize quotas for a filesystem
reset(1F) ..... reset the current form field to its default values
restore(1M) ..... incremental filesystem restore
rlog(1) ..... print log messages and other information about RCS files
rm(1): rm, rmdir ..... remove files or directories
rmdel(1) ..... remove a delta from an SCCS file
rmvopt(1M) ..... remove a menu option
run(1F) ..... run an executable
runacct(1M) ..... run daily accounting
sact(1) ..... print current SCCS file editing activity
sag(1G) ..... system activity graph
sar(1) ..... system activity reporter
sar(1M): sa1, sa2, sadc ..... system activity report package

savecore(1M) ..... save a core dump of the operating system
sccsdiff(1) ..... compare two versions of an SCCS file
sccstorcs(1) ..... build RCS file from SCCS file
sdiff(1) ..... side-by-side difference program
sed(1) ..... stream editor
set(1F): set, unset ..... set and unset environment variables in core or in files
setcolor(1F) ..... redefine or create a color
setmnt(1M) ..... establish mount table
setpgrp(1) ..... run command in its own process group
setup(1) ..... initialize system for first user
sh(1): sh, rsh ..... shell, the standard/restricted command programming language
shell(1F) ..... run a command using shell
showcfg(1M) ..... show the configuration of the machine
shutdown(1M) ..... shut down system, change system state
size(1) ..... see *C and Language Tools Binder*
sleep(1) ..... suspend execution for an interval
sort(1) ..... sort and/or merge files
spell(1): spell, hashmake, spellin, hashcheck ..... find spelling errors
spline(1G) ..... interpolate smooth curve
split(1) ..... split a file into pieces
starter(1) ..... information about the operating system for beginning users
stat(1G) ..... statistical network useful with graphical commands
strace(1M) ..... print STREAMS trace messages
strclean(1M) ..... STREAMS error logger cleanup program
strerr(1M) ..... STREAMS error logger daemon
strip(1) ..... see *C and Language Tools Binder*
stty(1) ..... set the options for a terminal
su(1M) ..... become superuser or another user
sum(1) ..... print checksum and block count of a file
swap(1M) ..... swap administrative interface
sweepon(1M): sweepon, sweepoff ..... enable/disable patrol seeks on DCC disks
sync(1M) ..... update the superblock
sysadm(1) ..... menu interface to do system administration
tabs(1) ..... set tabs on a terminal
tail(1) ..... deliver the last part of a file
tar(1) ..... tape file archiver
tasktbl(1M) ..... create security table in shared memory
tee(1) ..... pipe fitting
test(1) ..... condition evaluation command
tfipc(1M) ..... interprocess communication reporter
tfmenush(1M) ..... assign menu system as user login shell
tfnotify(1M) ..... send completion message
tic(1M) ..... terminfo compiler
time(1): time, ptime ..... time a command
timex(1) ..... time a command; report process data and system activity
toc(1G): dtoc, ttoc, vtoc ..... graphical table of contents routines
touch(1) ..... update access and modification times of a file

tplot(1G) ..... graphics filters
tput(1) ..... initialize a terminal or query terminfo database
tr(1) ..... translate characters
true(1): true, false ..... provide truth values
tsort(1) ..... topological sort
tty(1) ..... get the name of the terminal
ttydevname(1) ..... get the device associated with a terminal or the terminal associated with a device
tunefs(1M) ..... tune up an existing filesystem
uadmin(1M) ..... administrative control
umask(1) ..... set file-creation mode mask
uname(1) ..... print name of current system
unget(1) ..... undo a previous get of an SCCS file
uniq(1) ..... report repeated lines in a file
units(1) ..... conversion program
uptime(1) ..... show how long system has been up
usage(1) ..... retrieve a command description and usage examples
uucheck(1M) ..... check the uucp directories and permissions file
uucico(1M) ..... file transport program for the uucp system
uucleanup(1M) ..... uucp spool directory clean-up
uucp(1C): uucp, uulog, uuname ..... UNIX-to-UNIX system copy
uugetty(1M) ..... set terminal type, modes, speed, and line discipline
uusched(1M) ..... the scheduler for the uucp file transport program
uustat(1C) ..... uucp status inquiry and job control
uuto(1C): uuto, uupick ..... public UNIX-to-UNIX system file copy
uutry(1M) ..... try to contact remote system with debugging on
uux(1C) ..... UNIX-to-UNIX system command execution
uuxqt(1M) ..... execute remote command requests
val(1) ..... validate SCCS file
vc(1) ..... version control
vi(1) ..... screen-oriented (visual) display editor based on ex
vmtune(1M) ..... tune system-wide virtual-memory parameters
volcopy(1M) ..... make literal copy of filesystem
vsig(1F) ..... synchronize a co-process with the controlling FMLI object
wait(1) ..... await completion of process
wall(1) ..... write to all users
wc(1) ..... word count
what(1) ..... identify SCCS files
whatis(1) ..... describe what a command is
who(1) ..... who is on the system
whodo(1M) ..... who is doing what
write(1) ..... write to another user
xargs(1) ..... construct argument list(s) and execute command
yacc(1) ..... yet another compiler-compiler

2 System Calls

intro(2) ..... introduction to system calls and error numbers
access(2) ..... determine accessibility of a file
acct(2) ..... enable or disable process accounting
alarm(2) ..... set a process alarm clock
brk(2): brk, sbrk ..... change data segment space allocation
chdir(2) ..... change working directory
chmod(2) ..... change mode of file
chown(2) ..... change owner and group of a file
chroot(2) ..... change root directory
close(2) ..... close a file descriptor
creat(2) ..... create a new file or rewrite an existing one
dup(2) ..... duplicate an open file descriptor
dup2(2) ..... duplicate an open file descriptor
exec(2): execl, execv, execle,
execve, execlp, execvp ..... execute a file
exit(2): exit, _exit ..... terminate process
fcntl(2) ..... file control
fork(2) ..... create a new process
getdents(2) ..... read directory entries and put in a file system independent format
getgroups(2) ..... get supplementary group IDs
getmsg(2) ..... get next message off a stream
getpid(2): getpid, getpgrp, getppid .... get process, process group, and parent process IDs
getuid(2): getuid, geteuid, getgid,
getegid ..... get real user, effective user, real group, and effective group IDs
ioctl(2) ..... control device
kill(2) ..... send a signal to a process or a group of processes
link(2) ..... link to a file
lseek(2) ..... move read/write file pointer
mkdir(2) ..... make a directory
mknod(2): mknod ..... make a directory, or a special or ordinary file
mount(2) ..... mount a filesystem
msgctl(2) ..... message control operations
msgget(2) ..... get message queue
msgop(2) ..... message operations
nice(2) ..... change priority of a process
open(2) ..... open for reading or writing
pathconf(2): pathconf, fpathconf ..... get configurable pathname variables
pause(2) ..... suspend process until signal
pipe(2) ..... create an interprocess channel
plock(2) ..... lock process, text, or data in memory
poll(2) ..... STREAMS input/output multiplexing
profil(2) ..... execution time profile
ptrace(2): ptrace, mptrace ..... process trace facility
putmsg(2) ..... send a message on a stream
read(2) ..... read from file
rename(2) ..... change the name of a file
rmdir(2) ..... remove a directory

semctl(2) ..... semaphore control operations
semget(2) ..... get set of semaphores
semop(2) ..... semaphore operations
setpgid(2) ..... manipulate signal sets
setpgrp(2) ..... set process group ID
setsid(2) ..... create session and set process group ID
setuid(2): setuid, setgid ..... set user and group IDs
shmctl(2) ..... shared memory control operations
shmget(2) ..... get shared memory segment identifier
shmop(2): shmop, shmat, shmdt ..... shared memory operations
sigaction(2) ..... examine and change signal actions
sigemptyset(2): sigemptyset,
sigfillset, sigaddset, sigdelset,
sigismember ..... manipulate signal sets
signal(2) ..... specify what to do upon receipt of a signal
sigpending(2) ..... examine pending signals
sigprocmask(2) ..... examine and change blocked signals
sigset(2): sigset, sighold, sigrelse,
sigignore, sigpause ..... signal management
sigsuspend(2) ..... wait for a signal
stat(2): stat, lstat, fstat ..... get file status
statfs(2): statfs, fstatfs ..... get filesystem information
stime(2) ..... set time
stty(2): stty, gtty ..... set and get terminal state (defunct)
sync(2) ..... update superblock
syscall(2) ..... indirect system call
sysconf(2) ..... get configurable system variables
sysi86(2) ..... machine specific functions
tcgetattr(2): tcgetattr, tcsetattr ..... get/set terminal state
tcgetpgrp(2) ..... get foreground process group ID
tcsetpgrp(2) ..... set foreground process group ID
time(2) ..... get time
times(2) ..... get process and child process times
uadmin(2) ..... administrative control
ulimit(2) ..... get and set user limits
umask(2) ..... set and get file creation mask
umount(2) ..... unmount a filesystem
uname(2) ..... get name of current operating system
unlink(2) ..... remove directory entry
ustat(2) ..... get filesystem statistics
utime(2) ..... set file access and modification times
vfork(2) ..... spawn new process in a virtual memory efficient way
wait(2): wait, waitpid ..... wait for a child process to stop or terminate
write(2) ..... write on a file

2SEQ Sequent System Calls

intro(2SEQ) ..... introduction to Sequent system calls
bdflush(2SEQ) ..... update daemon
fvhangup(2SEQ) ..... virtually hangup the specified controlling terminal
get_process_stats(2SEQ) ..... get resource utilization information
getdtablemax(2SEQ) ..... get maximum in-use descriptor table index
getdtablesize(2SEQ) ..... get descriptor table size
getpagesize(2SEQ) ..... get system page size
mmap(2SEQ) ..... map an open file into the processs address space
munmap(2SEQ) ..... unmap part of the processs address space
proc_ctl(2SEQ) ..... control over various process attributes
quotactl(2SEQ) ..... manipulate disk quotas
readlink(2SEQ) ..... read value of a symbolic link
readv(2SEQ) ..... read vector from file
setdtablesize(2SEQ) ..... set descriptor table size
setgroups(2SEQ) ..... set group access list
sigcontext(2SEQ) ..... get signal context
sigstack(2SEQ) ..... set and/or get signal stack context
symlink(2SEQ) ..... make symbolic link to a file
tmp_affinity(2SEQ) ..... bind a process to a processor
tmp_ctl(2SEQ) ..... multi-processor control
truncate(2SEQ): truncate,
ftruncate ..... truncate a file to a specified length
vm_ctl(2SEQ) ..... examine and change virtual memory tuning parameters
writev(2SEQ) ..... write vector on a file

3 Subroutines

intro(3) ..... introduction to functions and libraries

3C C Library Routines

a64l(3C): a64l, l64a ..... convert between long integer and base-64 ASCII string
abort(3C) ..... generate an IOT fault
abs(3C) ..... return integer absolute value
bsearch(3C) ..... binary search a sorted table
clock(3C) ..... report CPU time used
conv(3C): toupper, tolower,
_toupper, _tolower, toascii ..... translate characters
crypt(3C): crypt, setkey, encrypt ..... generate hashing encryption
ctime(3C): ctime, localtime,
gmtime, asctime, cftime, ascftime,
mktime, tzset ..... convert date and time to string
ctype(3C): isdigit, isxdigit,
islower, isupper, isalpha, isalnum,
isspace, iscntrl, ispunct, isprint,
isgraph, isascii, tolower, toupper,
toascci, _tolower, _toupper,
setchrclass ..... character handling

dev_to_ttyname(3C): dev_to_ttyname, ttyname_to_dev ..... convert between major/minor numbers and tty names
dial(3C) ..... establish an out-going terminal line connection
drand48(3C): drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 ..... generate uniformly distributed pseudorandom numbers
ecvt(3C): ecvt, fcvt, gcvt ..... convert floating-point number to string
end(3C): end, etext, edata ..... last locations in program
fpgetround(3C): fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky ..... IEEE floating-point environment control
frexp(3C): frexp, ldexp, modf ..... manipulate parts of floating-point numbers
ftw(3C) ..... walk a file tree
getcwd(3C) ..... get pathname of current working directory
getenv(3C) ..... return value for environment name
getgrent(3C): getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent ..... get group file entry
getlogin(3C) ..... get login name
getopt(3C) ..... get option letter from argument vector
getpass(3C) ..... read a password
getpw(3C) ..... get name from UID
getpwent(3C): getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent ..... get password file entry
getut(3C): getutent, getutid, getutline, pututline, setutent, endutent, utmpname, getutpid ..... access utmp file entry
hsearch(3C): hsearch, hcreate, hdestroy ..... manage hash search tables
isnan(3C): isnand, isnanf ..... test for floating-point NaN (Not-A-Number)
l3tol(3C): l3tol, ltol3 ..... convert between 3-byte integers and long integers
lockf(3C) ..... record locking on files
lsearch(3C): lsearch, lfind ..... linear search and update
malloc(3C): malloc, free, realloc, calloc ..... main memory allocator
memory(3C): memccpy, memchr, memcmp, memcpy, memset ..... memory operations
mkfifo(3C) ..... make a FIFO special file
mktemp(3C) ..... make a unique filename
monitor(3C) ..... prepare execution profile
nlist(3C) ..... get entries from name list
perror(3C): perror, errno, sys_errlist, sys_nerr ..... system error messages
putenv(3C) ..... change or add value to environment
putpwent(3C) ..... write password file entry
qsort(3C) ..... quicker sort
rand(3C): rand, srand ..... simple random-number generator
setjmp(3C): setjmp, longjmp ..... non-local goto
setlocale(3C) ..... set programs location

sigsetjmp(3C): sigsetjmp,
siglongjmp ..... non-local jumps
sleep(3C) ..... suspend execution for interval
ssignal(3C): ssignal, gsignal ..... software signals
stdipc(3C) ..... standard interprocess communication package
strftime(3C) ..... format time information
string(3C): strcat, strdup, strncat,
strcmp, strncmp, strcpy, strncpy,
strlen, strchr, strrchr, strpbrk,
strspn, strcspn, strtok ..... string operations
strstr(3C) ..... find first occurrence of character
strtod(3C): strtod, atof ..... convert string to double-precision number
strtol(3C): strtol, atol, atoi ..... convert string to integer
swab(3C) ..... swap bytes
tsearch(3C): tsearch, tfind,
tdelete, twalk ..... manage binary search trees
ttyname(3C): ttyname, isatty ..... find name of a terminal
ttyslot(3C) ..... find the slot in the utmp file of the current user

3M Mathematical Library Routines

bessel(3M): j0, j1, jn, y0, y1, yn ..... Bessel functions
erf(3M): erf, erfc ..... error function and complementary error function
exp(3M): exp, log, log10, pow, sqrt .... exponential, logarithm, power, square root functions
floor(3M): floor, ceil, fmod, fabs ..... floor, ceiling, remainder, absolute value functions
gamma(3M) ..... log gamma function
hypot(3M) ..... Euclidean distance function
matherr(3M) ..... error-handling function
sinh(3M): sinh, cosh, tanh ..... hyperbolic functions
trig(3M): sin, cos, tan, asin, acos,
atan, atan2 ..... trigonometric functions

3N Networking Support Utilities

t_accept(3N) ..... accept a connect request
t_alloc(3N) ..... allocate a library structure
t_bind(3N) ..... bind an address to a transport endpoint
t_close(3N) ..... close a transport endpoint
t_connect(3N) ..... establish a connection with another transport user
t_error(3N) ..... produce error message
t_free(3N) ..... free a library structure
t_getinfo(3N) ..... get protocol-specific service information
t_getstate(3N) ..... get the current state
t_listen(3N) ..... listen for a connect request
t_look(3N) ..... look at the current event on a transport endpoint
t_open(3N) ..... establish a transport endpoint
t_optmgmt(3N) ..... manage options for a transport endpoint
t_rcv(3N) ..... receive data or expedited data sent over a connection
t_rcvconnect(3N) ..... receive the confirmation from a connect request
t_rcvdis(3N) ..... retrieve information from disconnect
t_rcvrel(3N) ..... acknowledge receipt of an orderly release indication

t_rcvudata(3N) ..... receive a data unit
t_rcvuderr(3N) ..... receive a unit data error indication
t_snd(3N) ..... send data or expedited data over a connection
t_snddis(3N) ..... send user-initiated disconnect request
t_sndrel(3N) ..... initiate an orderly release
t_sndudata(3N) ..... send a data unit
t_sync(3N) ..... synchronize transport library
t_unbind(3N) ..... disable a transport endpoint

3PPS Sequent Parallel Programming Library

intro(3PPS) ..... introduction to Parallel Programming Library
brk(3PPS): brk, sbrk ..... change private data segment size
cpus_online(3PPS) ..... returns the number of CPUs on-line
m_fork(3PPS) ..... execute a subprogram in parallel
m_get_myid(3PPS) ..... return process identification
m_get_numprocs(3PPS) ..... return number of child processes
m_kill_procs(3PPS) ..... kill child processes
m_lock(3PPS): m_lock, m_unlock ..... lock, unlock locks
m_next(3PPS) ..... increment global counter
m_park_procs(3PPS): m_park_procs,
m_rele_procs ..... suspend and resume child process execution
m_set_procs(3PPS) ..... set number of child processes
m_single(3PPS): m_single,
m_multi ..... mark single-process code section
m_sync(3PPS) ..... check in at barrier
s_lock(3PPS): s_init_lock, s_lock,
s_clock, s_unlock ..... initialize, lock, unlock locks
s_wait_barrier(3PPS): s_init_barrier,
s_wait_barrier ..... initialize barrier, wait at barrier
shbrk(3PPS): shbrk, shsbrk ..... change shared data segment size
shmalloc(3PPS): shmalloc,
shrealloc, shfree ..... shared memory allocator

3S Standard I/O Library Routines

ctermid(3S) ..... generate filename for terminal
cuserid(3S) ..... get character login name of the user
fclose(3S): fclose, fflush ..... close or flush a stream
ferror(3S): ferror, feof, clearerr,
fileno ..... stream status inquiries
fopen(3S): fopen, freopen, fdopen ..... open a stream
fread(3S): fread, fwrite ..... binary input/output
fseek(3S): fseek, rewind, ftell ..... reposition a file pointer in a stream
getc(3S): getc, getchar, fgetc, getw .... get character or word from a stream
gets(3S): gets, fgets ..... get a string from a stream
popen(3S): popen, pclose ..... initiate pipe to/from a process
printf(3S): printf, fprintf, sprintf ..... print formatted output

putc(3S): putc, putchar, fputc,
putw ..... put character or word on a stream
puts(3S): puts, fputs ..... put a string on a stream
scanf(3S): scanf, fscanf, sscanf ..... convert formatted input
setbuf(3S): setbuf, setvbuf ..... assign buffering to a stream
stdio(3S) ..... standard buffered input/output package
system(3S) ..... issue a shell command
tmpfile(3S) ..... create a temporary file
tmpnam(3S): tmpnam, tempnam ..... create a name for a temporary file
ungetc(3S) ..... push character back into input stream
vprintf(3S): vprintf, vfprintf,
vsprintf ..... print formatted output of a varargs argument list

3SEQ Sequent I/O Library Routines

dis386(3SEQ) ..... disassemble Series 386 instructions
getgeombyname(3SEQ) ..... get disk geometry by name
getmntent(3SEQ): addmntent,
endmntent, getmntent,
hasmntopt, setmntent ..... get file system descriptor file entry
getpseudotty(3SEQ) ..... get pseudo terminals
getscsiinfo(3SEQ): getscsiinfo,
getscsimatch ..... get scsiinfo structure
getusclk(3SEQ): getusclk,
usclk_init ..... get/init microsecond clock
getzdinfo(3SEQ): getzdinfo,
getzdinfobyname ..... get zdinfo structure
initgroups(3SEQ) ..... initialize group access list
nblocks(3SEQ) ..... calculate number of blocks and indirect blocks
offline_all(3SEQ) ..... take all but one active processor off line
read_constab(3SEQ) ..... read constab entries into a structure
valloc(3SEQ) ..... aligned memory allocator

3X Special Routines

assert(3X) ..... verify program assertion
crypt(3X) ..... password and file encryption functions
curses(3X) ..... terminal screen handling and optimization package
directory(3X): opendir, readdir,
telldir, seekdir, rewinddir, closedir .... directory operations
getspent(3X): getspent, getspnam,
setspent, endspent, fgetspent,
lckpwdf, ulckpwdf ..... get shadow password file entry
ldahread(3X) ..... read the archive header of a member of an archive file
ldclose(3X): ldclose, ldaclose ..... close a common object file
ldfhread(3X) ..... read the file header of a common object file
ldgetname(3X) ..... retrieve symbol name for common object file symbol table entry

ldlread(3X): ldlread, ldlinit, ldlitem .. manipulate line number entries of a common object file function
ldlseek(3X): ldlseek, ldnlseek ..... seek to line number entries of a section of a common object file
ldohseek(3X) ..... seek to the optional file header of a common object file
ldopen(3X): ldopen, ldaopen ..... open a common object file for reading
ldrseek(3X): ldrseek, ldnrseek ..... seek to relocation entries of a section of a common object file
ldshread(3X): ldshread, ldnshread .... read an indexednamed section header of a common object file
ldsseek(3X): ldsseek, ldnsseek ..... seek to an indexednamed section of a common object file
ldtbindex(3X) ..... compute the index of a symbol table entry of a common object file
ldtbread(3X) ..... read an indexed symbol table entry of a common object file
ldtbseek(3X) ..... seek to the symbol table of a common object file
logname(3X) ..... return login name of user
malloc(3X): malloc, free, realloc, calloc, mallopt, mallinfo ..... fast main memory allocator
ndbm(3X): dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush ..... database subroutines
plot(3X) ..... graphics interface subroutines
putspent(3X) ..... write shadow password file entry
regcmp(3X): regcmp, regex ..... compile and execute regular expression
sputl(3X): sputl, sgetl ..... access long integer data in a machine-independent fashion
utadduser(3X): ut_add_user, ut_delete_user ..... add/delete users in utmp file

4 File Formats

intro(4) ..... introduction to file formats
a.out(4) ..... common assembler and link editor output
acct(4) ..... per-process accounting file format
addusr_def(4) ..... add user defaults
alias(4) ..... alias file for FACE
ar(4) ..... see *C and Language Tools Binder*
backup_codes(4) ..... backup cycle code definitions
backup_codes1(4) ..... backup subcode definitions
backup_dev(4) ..... backup devices
backup_info(4) ..... backup set definitions
cftime(4) ..... language specific strings
checklist(4) ..... list of filesystems processed by fsck and ncheck
constab(4) ..... constructed devices table
core(4) ..... format of core image file
cpio(4) ..... format of cpio archive
cronlog(4) ..... cron history file
crontabs(4) ..... clock daemon file
devtab(4) ..... virtual device table

dir(4) ..... format of directories
dirent(4) ..... file system independent directory entry
diskinfo(4) ..... disk device characteristic database
dump(4): dump, dumpdates ..... incremental dump format
env(4): .environ, .pref, .variables ..... FACE files
errorlog(4) ..... internal error log file
errormsg(4) ..... error message text file
filehdr(4) ..... file header for common object files
fs(4): fs, inode ..... format of file system volume
fspec(4) ..... format specification in text files
fstab(4) ..... file-system-table
gettydefs(4) ..... speed and terminal settings used by getty
gps(4): gps ..... graphical primitive string, format of graphical files
group(4) ..... group file
helplog(4) ..... monitor help actions
init.d(4) ..... system level transition files directory
inittab(4) ..... script for the init process
issue(4) ..... issue identification file
ldfcn(4) ..... common object file access routines
limits(4) ..... file header for implementation-specific constants
linenum(4) ..... line number entries in a common object file
loginlog(4) ..... log of failed login attempts
lptypes(4) ..... table of printer types supported by the menu system
mnttab(4) ..... mounted filesystem table
mod_config(4) ..... format of a module configuration file
motd(4) ..... message-of-the-day file
ott(4) ..... files that hold object architecture information
passwd(4) ..... password file
plot(4) ..... graphics interface
product(4) ..... installed products file
profile(4) ..... setting up an environment at login time
proto(4) ..... prototype job file for at
rc(4) ..... reboot and shutdown command scripts
rcsfile(4) ..... format of RCS file
reloc(4) ..... relocation information for a common object file
sccsfile(4) ..... format of SCCS file
scnhdr(4) ..... section header for a common object file
scr_dump(4) ..... format of curses screen image file
shadow(4) ..... restricted password file
shells(4) ..... shell table
shutdown(4) ..... system shutdown information file
spellhist(4) ..... unmatched spelling words file
sulog(4) ..... log of substitute user events
syms(4) ..... common object file symbol table format
sys_config(4) ..... format of a system configuration file
syskeymap(4) ..... identify keys to internal menu system functions
task(4) ..... global security table
term(4) ..... format of compiled term file
termcap(4) ..... terminal capability data base
terminfo(4) ..... terminal capability database

timezone(4) ..... set default system time zone
unistd(4) ..... file header for symbolic constants
utmp(4): utmp, wtmp ..... utmp and wtmp entry formats
uucp(4) ..... networking support files

5 Miscellaneous

intro(5) ..... introduction to miscellany
ascii(5) ..... map of ASCII character set
environ(5) ..... user environment
fcntl(5) ..... file control options
math(5) ..... math functions and constants
prof(5) ..... profile within a function
regexp(5) ..... regular expression compile and match routines
stat(5) ..... data returned by stat system call
term(5) ..... conventional names for terminals
types(5) ..... primitive system data types
values(5) ..... machine-dependent values
varargs(5) ..... handle variable argument list

7 Special Files

intro(7) ..... introduction to special files
clone(7) ..... open any minor device on a STREAMS driver
cmpt(7) ..... compatibility partitioning driver
cn(7) ..... pseudoconsole driver
consmem(7): consmem, sm, ss ..... console memory drivers
console(7) ..... console interface
kmmem(7): kMBmem, kMWmem ..... MULITBUS memory
ld0(7) ..... STREAMS tty line discipline
log(7) ..... interface to STREAMS error logging and event tracing
lp(7) ..... Systech parallel line printer interface
lpmod(7) ..... parallel printer canonical processing module
mbad(7) ..... MULITBUS adapter
mem(7): mem, kmem ..... core memory
mtio(7) ..... UNIX magnetic tape interface
null(7) ..... the null file
pmap(7) ..... map physical addresses into processs virtual address space
sa(7) ..... devices administered by System Administration
sced(7) ..... SCSI/Ethernet/Diagnostics board
sd(7) ..... SCSI disk controller
sp(7) ..... STREAMS pipe device
ssm(7) ..... System Services Module
st(7) ..... Systech MTI-800/1600/1650 terminal multiplexor driver
streamio(7) ..... STREAMS ioctl commands
streampty(7) ..... STREAMS pseudo terminal driver
streamtty(7) ..... Generic STREAMS tty driver interface
termio(7) ..... general terminal interface
tg(7) ..... SCSI 1/2-inch magnetic tape interface

timod(7) ..... Transport Interface cooperating STREAMS module
tirdwr(7) ..... Transport Interface read/write interface STREAMS module
tm(7) ..... SCSI tape controller
ts(7) ..... SCSI tape controller
tty(7) ..... controlling terminal (gentty) interface
usclk(7) ..... 32-bit microsecond counter
vtoc(7) ..... VTOC partitioning driver
wd(7) ..... SSM SCSI disk controller
wp(7) ..... SSM parallel line printer interface
xs(7) ..... Xylogics 781 terminal multiplexor driver
xt(7) ..... Xylogics 472 MULITBUS magnetic tape interface
zd(7) ..... dual-channel disk controller

8 Stand-Alone Utilities

intro(8) ..... introduction to system standalone utilities
ccsformat(8) ..... standalone embedded SCSI disk formatter
boot(8) ..... run a standalone program or bring up the operating system
copy(8) ..... perform a standalone data transfer
dump(8) ..... standalone memory dump
prtvtoc(8) ..... print the VTOC of a standalone disk device
zdformat(8) ..... standalone disk formatter for dual-channel disk controller

Permuted Index

st: Systech MTI-800/1600/ 1650 terminal multiplexor driver.....st(7)
hpio: Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
of DASI 300 and 300s terminals 300, 300s: handle special functions.....300(1)
handle special functions of DASI 300 and 300s terminals.....300(1)
DASI 300 and 300s terminals 300, 300s: handle special functions of.....300(1)
special functions of DASI 300 and 300s terminals.....300(1)
usclk: 32-bit microsecond counter.....usclk(7)
dis386: disassemble Series 386 instructions.....dis386(3SEQ)
DASI 450 terminal 450: handle special functions of the.....450(1)
handle special functions of the DASI 450 terminal.....450(1)
xt: Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
between long integer and base- 64 ASCII string.....a64l(3C)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
integer and base-64 ASCII string a64l, l64a: convert between long.....a64l(3C)
abort: generate an IOT fault.....abort(3C)
abs: return integer absolute value.....abs(3C)
abs: return integer absolute value.....abs(3C)
fabs: floor, ceiling, remainder, absolute value functions.....floor(3M)
t_accept: accept a connect request.....t_accept(3N)
requests accept, reject: allow or prevent LP.....accept(1M)
utime: set file access and modification times.....utime(2)
touch: update access and modification times of a file.....touch(1)
file access: determine accessibility of a.....access(2)
initgroups: initialize group access list.....initgroups(3SEQ)
setgroups: set group access list.....setgroups(2SEQ)
machine-independent/ sputl, sgetl: access long integer data in a.....sputl(3X)
ldfcn: common object file access routines.....ldfcn(4)
endutent, utmpname, getutpid: access utmp file entry.....getut(3C)
access: determine accessibility of a file.....access(2)
acct: enable or disable process accounting.....acct(2)
acctcon1, acctcon2: connect-time accounting.....acctcon(1M)
acctprc1, acctprc2: process accounting.....acctprc(1M)
turnacct: shell procedures for accounting.....acctsh(1M)
runacct: run daily accounting.....runacct(1M)
/accton, acctwtmp: overview of accounting and miscellaneous/.....acct(1M)
of accounting and miscellaneous accounting commands.....acct(1M)
diskusg: generate disk accounting data by user ID.....diskusg(1M)
acct: per-process accounting file format.....acct(4)
acctcom: search and print process accounting files.....acctcom(1)
acctmerg: merge or add total accounting files.....acctmerg(1M)
command summary from per-process accounting records.....acctcms(1M)
fwtmp, wtmpfix: manipulate connect accounting records.....fwtmp(1M)
accounting acct: enable or disable process.....acct(2)
format acct: per-process accounting file.....acct(4)
per-process accounting records acctcms: command summary from.....acctcms(1M)
accounting files acctcom: search and print process.....acctcom(1)
accounting acctcon1, acctcon2: connect-time.....acctcon(1M)

acctcon1, acctcon2: connect-time accounting ..... acctcon(1M)
acctwtmp: overview of accounting/ acctdisk, acctdusg, accton, ... acct(1M)
of accounting and/ acctdisk, acctdusg, accton, acctwtmp: overview ..... acct(1M)
accounting files acctmerg: merge or add total ..... acctmerg(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and/ ..... acct(1M)
accounting acctprc1, acctprc2: process ..... acctprc(1M)
acctprc1, acctprc2: process accounting ..... acctprc(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and/ ..... acct(1M)
release indication t_rcvrel: acknowledge receipt of an orderly ..... t_rcvrel(3N)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric/ ..... trig(3M)
helplog: monitor help actions ..... helplog(4)
sigaction: examine and change signal actions ..... sigaction(2)
killall: kill all active processes ..... killall(1M)
offline_all: take all but one active processor off line ..... offline_all(3SEQ)
mbad: MULITBUS adapter ..... mbad(7)
addcmdopt: add a command option to a menu ..... addcmdopt(1M)
menu addformopt: add a form/SELECT window option to a ..... addformopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
addusr.def: add user defaults ..... addusr_def(4)
putenv: change or add value to environment ..... putenv(3C)
menu addcmdopt: add a command option to a ..... addcmdopt(1M)
ut_add_user, ut_delete_user: add/delete users in utmp file ..... utadduser(3X)
option to a menu addformopt: add a form/SELECT window ... addformopt(1M)
menu addmenuopt: add a menu option to a ..... addmenuopt(1M)
hasmntopt, setmntent: get file/ addmntent, endmntent, getmntent, ... getmntent(3SEQ)
map an open file into the processs address space ..... mmap(2SEQ)
munmap: unmap part of the processs address space ..... munmap(2SEQ)
addresses into processs virtual address space ..... pmap(7)
t_bind: bind an address to a transport endpoint ..... t_bind(3N)
pmap: map physical addresses into processs virtual address space ..... pmap(7)
addusr.def: add user defaults ..... addusr_def(4)
files admin: create and administer SCCS ..... admin(1)
print service lpfilter: administer filters used with the LP ..... lpfilter(1M)
print service lpforms: administer forms used with the LP ..... lpforms(1M)
admin: create and administer SCCS files ..... admin(1)
SA: devices administered by System Administration ..... sa(7)
SA: devices administered by System Administration ..... sa(7)
sysadm: menu interface to do system administration ..... sysadm(1)
uadmin: administrative control ..... uadmin(1M)
uadmin: administrative control ..... uadmin(2)
swap: swap administrative interface ..... swap(1M)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
/onproc: run a command with priority aging, PFF, swapping, or processor/ ..... noage(1M)
alarm: set a process alarm clock ..... alarm(2)
alarm: set a process alarm clock ..... alarm(2)
/display application specific alarms and/or the working indicator ..... indicator(1F)
pathalias: alias file for FACE ..... alias(4)
valloc: aligned memory allocator ..... valloc(3SEQ)
t_alloc: allocate a library structure ..... t_alloc(3N)
brk, sbrk: change data segment space allocation ..... brk(2)
free, realloc, calloc: main memory allocator ..... malloc(3C)

mallopt, mallinfo: fast main memory allocator.....malloc(3X)
shrealloc, shfree: shared memory allocator.....shmalloc(3PPS)
valloc: aligned memory allocator.....valloc(3SEQ)
editor output a.out: common assembler and link .....a.out(4)
to maintenance commands and application programs.....intro(1)
the working/ indicator: display application specific alarms and/or.....indicator(1F)
lookup apropos: locate commands by keyword.....apropos(1)
ar: archive file format.....see *C and Lang Tools Binder*
and libraries ar: maintain archives.....see *C and Lang Tools Binder*
bc: arbitrary-precision arithmetic language .....bc(1)
ott: files that hold object architecture information.....ott(4)
cpio: format of cpio archive .....cpio(4)
the archive header of a member of an archive file.....ldahread(3X)
ar: common archive file format .....see *C and Lang Tools Binder*
archive file ldahread: read the archive header of a member of an .....ldahread(3X)
2645A terminal tape file archiver.....hpio(1)
tar: tape file archiver .....tar(1)
cpio: copy file archives in and out.....cpio(1)
varargs: handle variable argument list .....varargs(5)
print formatted output of a varargs argument list .....vprintf(3S)
xargs: construct argument list(s) and execute command.....xargs(1)
getopt: get option letter from argument vector.....getopt(3C)
echo: echo arguments.....echo(1)
expr: evaluate arguments as an expression .....expr(1)
message: puts its arguments on message line .....message(1F)
bc: arbitrary-precision arithmetic language .....bc(1)
asa: interpret ASA carriage control characters.....asa(1)
characters asa: interpret ASA carriage control.....asa(1)
/localtime, gmtime, asctime, cftime, ascftime, mktime, tzset: convert/.....ctime(3C)
ascii: map of ASCII character set .....ascii(5)
ascii: map of ASCII character set.....ascii(5)
between long integer and base-64 ASCII string.....a64l(3C)
ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset:/.....ctime(3C)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric/.....trig(3M)
as: common assembler.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
assert: verify program assertion.....assert(3X)
assert: verify program assertion.....assert(3X)
system commands assist: assistance using operating.....assist(1)
/program for generating/modifying ASSIST menus or command forms.....astgen(1)
assist: assistance using operating system commands .....assist(1)
generating/modifying ASSIST menus/ astgen: program for.....astgen(1)
later time at, batch: execute commands at a .....at(1)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric functions.....trig(3M)
/sin, cos, tan, asin, acos, atan, atan2: trigonometric functions.....trig(3M)
double-precision number strtod, atof: convert string to.....strtod(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
change login password and password attributes.....passwd(1)
control over various process attributes .....proc_ctl(2SEQ)
rcs: change RCS file attributes.....rcs(1)
dumpconf: dump kernel auto-configuration device table.....dumpconf(1M)

language awk: pattern scanning and processing.....awk(1)
backup.codes: backup cycle code definitions.....backup_codes(4)
backup.dev: backup devices.....backup_dev(4)
ckbupscd: check filesystem backup schedule .....ckbupscd(1M)
backup.info: backup set definitions.....backup_info(4)
backup.codes1: backup subcode definitions.....backup_codes1(4)
definitions backup.codes: backup cycle code .....backup_codes(4)
definitions backup.codes1: backup subcode.....backup_codes1(4)
backup.dev: backup devices .....backup_dev(4)
backup.info: backup set definitions.....backup_info(4)
banner: make posters.....banner(1)
m_sync: check in at barrier.....m_sync(3PPS)
initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
/s_wait_barrier: initialize barrier, wait at barrier .....s_wait_barrier(3PPS)
of pathnames basename, dirname: deliver portions.....basename(1)
at, batch: execute commands at a later time.....at(1)
language bc: arbitrary-precision arithmetic .....bc(1)
procedures bcheckrc: system initialization.....bcheckrc(1M)
bdflush: update daemon.....bdflush(1M)
bdflush: update daemon.....bdflush(2SEQ)
bdiff: big diff.....bdiff(1)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions .....bessel(3M)
Bessel functions bessel: j0, j1, jn, y0, y1, yn: .....bessel(3M)
bfs: big file scanner.....bfs(1)
fread, fwrite: binary input/output.....fread(3S)
bp: binary patcher.....bp(1M)
bsearch: binary search a sorted table.....bsearch(3C)
tfind, tdelete, twalk: manage binary search trees.....tsearch(3C)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
t_bind: bind an address to a transport endpoint.....t_bind(3N)
sum: print checksum and block count of a file.....sum(1)
prtvtoc: print the VTOC of a block device .....prtvtoc(1M)
sigprocmask: examine and change blocked signals.....sigprocmask(2)
number of blocks and indirect blocks.....nblocks(3SEQ)
nblocks: calculate number of blocks and indirect blocks.....nblocks(3SEQ)
df: report number of free disk blocks and inodes.....df(1M)
consistency check based on the BOM.....bomverify(1M)
consistency check based on the BOM bomverify: perform an installation.....bomverify(1M)
bootflags: display/change system boot flags.....bootflags(1M)
installboot: installs a boot program on a disk.....installboot(1M)
bring up the operating system boot: run a standalone program or.....boot(8)
boot flags bootflags: display/change system .....bootflags(1M)
bp: binary patcher.....bp(1M)
allocation brk, sbrk: change data segment space.....brk(2)
segment size brk, sbrk: change private data.....brk(3PPS)
table bsearch: binary search a sorted .....bsearch(3C)
stdio: standard buffered input/output package .....stdio(3S)
setbuf, setvbuf: assign buffering to a stream.....setbuf(3S)
mailbug: submit a bug report or enhancement request .....mailbug(1)
devbuild, devdestroy: build and destroy virtual devices .....devbuild(1M)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
mknod: build special file.....mknod(1M)

config: build system configuration files.....config(1M)
swab: swap bytes.....swab(3C)
cc: C compiler.....see *C and Lang Tools Binder*
cflow: generate C flowgraph.....see *C and Lang Tools Binder*
cpp: the C language preprocessor.....see *C and Lang Tools Binder*
cb: C program beautifier.....see *C and Lang Tools Binder*
lint: a C program checker.....see *C and Lang Tools Binder*
cxref: generate a C program cross-reference.....see *C and Lang Tools Binder*
ctrace: C program debugger.....see *C and Lang Tools Binder*
list: produce C source listing.....see *C and Lang Tools Binder*
cal: print calendar.....cal(1)
indirect blocks nblocks: calculate number of blocks and.....nblocks(3SEQ)
dc: desk calculator.....dc(1)
cal: print calendar.....cal(1)
calendar: reminder service.....calendar(1)
stat: data returned by stat system call.....stat(5)
syscall: indirect system call.....syscall(2)
cu: call another UNIX system.....cu(1C)
malloc, free, realloc, calloc: main memory allocator.....malloc(3C)
malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory/.....malloc(3X)
introduction to Sequent system calls.....intro(2SEQ)
intro: introduction to system calls and error numbers.....intro(2)
lp, cancel: send/ cancel requests to the LP print service.....lp(1)
LP print service lp, cancel: send/cancel requests to the.....lp(1)
lpmod: parallel printer canonical processing module.....lpmod(7)
description into a terminfo/ captoinfo: convert a termcap.....captoinfo(1M)
asa: interpret ASA carriage control characters.....asa(1)
cat: concatenate and print files.....cat(1)
cb: C program beautifier.....see *C and Lang Tools Binder*
gencc: create a front-end to the cc command.....gencc(1M)
cc: C compiler.....see *C and Lang Tools Binder*
disk formatter CCSformat: standalone embedded SCSI.....ccsformat(8)
cd: change working directory.....cd(1)
an SCCS delta cdc: change the delta commentary of.....cdc(1)
remainder, absolute value/ floor, ceil, fmod, fabs: floor, ceiling,.....floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/.....floor(3M)
cflow: generate C flowgraph...see *C and Lang Tools Binder*
ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset/.....ctime(3C)
cftime: language specific strings.....cftime(4)
sigprocmask: examine and change blocked signals.....sigprocmask(2)
brk, sbrk: change data segment space allocation.....brk(2)
attributes passwd: change login password and password.....passwd(1)
chmod: change mode.....chmod(1)
chmod: change mode of file.....chmod(2)
putenv: change or add value to environment.....putenv(3C)
chown: change owner and group of a file.....chown(2)
chown, chgrp: change owner or group.....chown(1)
nice: change priority of a process.....nice(2)
brk, sbrk: change private data segment size.....brk(3PPS)
rcs: change RCS file attributes.....rcs(1)
chroot: change root directory.....chroot(2)
chroot: change root directory for a command.....chroot(1M)

shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
sigaction: examine and change signal actions ..... sigaction(2)
bootflags: display/ change system boot flags ..... bootflags(1M)
shutdown: shut down system, change system state ..... shutdown(1M)
cdc: change the delta commentary of an SCCS delta ..... cdc(1)
newform: change the format of a text file ..... newform(1)
helpadm: change the Help Facility database ..... helpadm(1M)
rename: change the name of a file ..... rename(2)
delta: make a delta (change) to an SCCS file ..... delta(1)
parameters vm_ctl: examine and change virtual memory tuning ..... vm_ctl(2SEQ)
cd: change working directory ..... cd(1)
chdir: change working directory ..... chdir(2)
pipe: create an interprocess channel ..... pipe(2)
zd: dual- channel disk controller ..... zd(7)
standalone disk formatter for dual- channel disk controller ..... zdformat(8)
strstr: find first occurrence of character ..... strstr(3C)
ungetc: push character back into input stream ..... ungetc(3S)
conversion tables chrtbl: generate character classification and ..... chrtbl(1M)
_tolower, _toupper, setchrclass: character handling ..... ctype(3C)
cuserid: get character login name of the user ..... cuserid(3S)
getc, getchar, fgetc, getw: get character or word from a stream ..... getc(3S)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
ascii: map of ASCII character set ..... ascii(5)
fgrep: search a file for a character string ..... fgrep(1)
diskinfo: disk device characteristic database ..... diskinfo(4)
asa: interpret ASA carriage control characters ..... asa(1)
_tolower, toascii: translate characters ..... conv(3C)
tr: translate characters ..... tr(1)
lastlogin, monacct, nulladm,/ chargefee, ckpacct, dodisk, ... acctsh(1M)
chdir: change working directory ..... chdir(2)
filesystem directory consistency check ..... dcheck(1M)
filesystem storage consistency check ..... icheck(1M)
checkfsys: check a filesystem ..... checkfsys(1M)
fsck: filesystem consistency check and interactive repair ..... fsck(1M)
perform an installation consistency check based on the BOM ..... bomverify(1M)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
m_sync: check in at barrier ..... m_sync(3PPS)
ci: check in RCS revisions ..... ci(1)
co: check out RCS revisions ..... co(1)
permissions file uucheck: check the uucp directories and ..... uucheck(1M)
eqn syntax. checkeq: check documents for correct ..... checkeq(1)
checkfsys: check a filesystem ..... checkfsys(1M)
processed by fsck and ncheck checklist: list of filesystems ..... checklist(4)
sum: print checksum and block count of a file ..... sum(1)
chown, chgrp: change owner or group ..... chown(1)
/m_rele_procs: suspend and resume child process execution ..... m_park_procs(3PPS)
times: get process and child process times ..... times(2)
wait, waitpid: wait for a child process to stop or terminate ..... wait(2)
m_get_numprocs: return number of child processes ..... m_get_numprocs(3PPS)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
m_set_procs: set number of child processes ..... m_set_procs(3PPS)

chmod: change mode .....chmod(1)
chmod: change mode of file.....chmod(2)
file chown: change owner and group of a.....chown(2)
chown, chgrp: change owner or group .....chown(1)
chroot: change root directory.....chroot(2)
command chroot: change root directory for a.....chroot(1M)
classification and conversion/ chrtbl: generate character .....chrtbl(1M)
ci: check in RCS revisions.....ci(1)
schedule ckbupscd: check filesystem backup.....ckbupscd(1M)
nulladm, prctmp,/ chargefee, ckpacct, dodisk, lastlogin, monacct, ...acctsh(1M)
chrtbl: generate character classification and conversion tables.....chrtbl(1M)
clri: clear inode.....clri(1M)
ferror, feof, clearerr, fileno: stream status inquiries .....ferror(3S)
a shell (command interpreter) with C-like syntax.....csh(1)
alarm: set a process alarm clock.....alarm(2)
usclk_init: get/init microsecond clock.....getusclk(3SEQ)
cron: clock daemon.....cron(1M)
crontabs: clock daemon file.....crontabs(4)
clock: report CPU time used .....clock(3C)
STREAMS driver clone: open any minor device on a .....clone(7)
ldclose, ldaclose: close a common object file.....ldclose(3X)
close: close a file descriptor.....close(2)
t_close: close a transport endpoint.....t_close(3N)
close: close a file descriptor .....close(2)
fclose, fflush: close or flush a stream.....fclose(3S)
/telldir, seekdir, rewinddir, closedir: directory operations.....directory(3X)
clri: clear inode .....clri(1M)
cmp: compare two files.....cmp(1)
driver cmpt: compatibility partitioning.....cmpt(7)
cn: pseudoconsole driver .....cn(7)
co: check out RCS revisions.....co(1)
communicate to a/ cocreate, cosend, cocheck, coreceive, codestroy: .....coproc(1F)
coreceive, codestroy: communicate/ cocreate, cosend, cocheck,.....coproc(1F)
backup.codes: backup cycle code definitions.....backup_codes(4)
m_multi: mark single-process code section.....m_single(3PPS)
/cosend, cocheck, coreceive, codestroy: communicate to a process .....coproc(1F)
col: filter reverse line-feeds.....col(1)
to form error log dmesg: collect system diagnostic messages.....dmesg(1M)
setcolor: redefine or create a color.....setcolor(1F)
comb: combine SCCS deltas.....comb(1)
comb: combine SCCS deltas.....comb(1)
to two sorted files comm: select or reject lines common.....comm(1)
mcs: manipulate the object file comment section.....mcs(1)
cdc: change the delta commentary of an SCCS delta .....cdc(1)
a.out: common assembler and link editor output.....a.out(4)
ldclose, ldaclose: close a common object file.....ldclose(3X)
ldfhread: read the file header of a common object file.....ldfhread(3X)
number entries of a section of a common object file .....ldlseek(3X)
to the optional file header of a common object file.....ldohseek(3X)
relocation entries of a section of a common object file.....ldrseek(3X)
indexednamed section header of a common object file.....ldshread(3X)
to an indexednamed section of a common object file.....ldsseek(3X)

index of a symbol table entry of a common object file ..... ldtbindex(3X)
an indexed symbol table entry of a common object file ..... ldtbread(3X)
seek to the symbol table of a common object file ..... ldtbseek(3X)
linenum: line number entries in a common object file ..... linenum(4)
reloc: relocation information for a common object file ..... reloc(4)
scnhdr: section header for a common object file ..... scnhdr(4)
ldfcn: common object file access routines ..... ldfcn(4)
ldopen, ldaopen: open a common object file for reading ..... ldopen(3X)
/manipulate line number entries of a common object file function ..... ldlread(3X)
ldgetname: retrieve symbol name for common object file symbol table/ ..... ldgetname(3X)
syms: common object file symbol table format ..... syms(4)
filehdr: file header for common object files ..... filehdr(4)
symbols glossary: definitions of common operating system terms and ..... glossary(1)
comm: select or reject lines common to two sorted files ..... comm(1)
cocheck, coreceive, codestroy: communicate to a process ..... coproc(1F)
ipcs: report inter-process communication facilities status ..... ipcs(1)
stdipc: ftok: standard interprocess communication package ..... stdipc(3C)
tfipc: interprocess communication reporter ..... tfipc(1M)
descriptions infocmp: compare or print out terminfo ..... infocmp(1M)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
cmp: compare two files ..... cmp(1)
sccsdiff: compare two versions of an SCCS file ..... sccsdiff(1)
diff: differential file comparison ..... diff(1)
diff3: 3-way differential file comparison ..... diff3(1)
dircmp: directory comparison ..... dircmp(1)
cmpt: compatibility partitioning driver ..... cmpt(7)
regcmp: regular expression compile ..... regcmp(1)
regcmp, regex: compile and execute regular expression ..... regcmp(3X)
regexp: regular expression compile and match routines ..... regexp(5)
term: format of compiled term file ..... term(4)
cc: C compiler ..... see *C and Lang Tools Binder*
ec: forms compiler ..... ec(1M)
tic: terminfo compiler ..... tic(1M)
yacc: yet another compiler- compiler ..... yacc(1)
yacc: yet another compiler-compiler ..... yacc(1)
erf, erfc: error function and complementary error function ..... erf(3M)
cprs: compress an object file ..... see *C and Lang Tools Binder*
compress, uncompress, zcat: compress and expand data ..... compress(1)
pack, pcat, unpack: compress and expand files ..... pack(1)
and expand data compress, uncompress, zcat: compress ..... compress(1)
entry of a common object/ ldtbindex: compute the index of a symbol table ..... ldtbindex(3X)
cat: concatenate and print files ..... cat(1)
test: condition evaluation command ..... test(1)
files config: build system configuration ..... config(1M)
pathconf, fpathconf: get configurable pathname variables ..... pathconf(2)
sysconf: get configurable system variables ..... sysconf(2)
dumpconf: dump kernel auto- configuration device table ..... dumpconf(1M)
mod_config: format of a module configuration file ..... mod_config(4)
sys_config: format of a system configuration file ..... sys_config(4)
config: build system configuration files ..... config(1M)
showcfg: show the configuration of the machine ..... showcfg(1M)
lpadmin: configure the LP print service ..... lpadmin(1M)

t_rcvconnect: receive the — confirmation from a connect request ..... t_rcvconnect(3N)
fwtmp, wtmpfix: manipulate — connect accounting records ..... fwtmp(1M)
t_accept: accept a — connect request ..... t_accept(3N)
t_listen: listen for a — connect request ..... t_listen(3N)
receive the confirmation from a — connect request ..... t_rcvconnect(3N)
establish an out-going terminal line — connection ..... dial(3C)
data or expedited data sent over a — connection ..... t_rcv(3N)
send data or expedited data over a — connection ..... t_snd(3N)
t_connect: establish a — connection with another transport user ..... t_connect(3N)
acctcon1, acctcon2: — connect-time accounting ..... acctcon(1M)
dcheck: filesystem directory — consistency check ..... dcheck(1M)
icheck: filesystem storage — consistency check ..... icheck(1M)
fsck: filesystem — consistency check and interactive repair ..... fsck(1M)
bomverify: perform an installation — consistency check based on the BOM ..... bomverify(1M)
quotacheck: filesystem quota — consistency checker ..... quotacheck(1M)
drivers — consmem, sm, ss: console memory ..... consmem(7)
— console: console interface ..... console(7)
console: — console interface ..... console(7)
consmem, sm, ss: — console memory drivers ..... consmem(7)
— constab: constructed devices table ..... constab(4)
read_constab: read — constab entries into a structure ..... read_constab(3SEQ)
header for implementation-specific — constants ..... limits(4)
math: math functions and — constants ..... math(5)
unistd: file header for symbolic — constants ..... unistd(4)
constab: — constructed devices table ..... constab(4)
deroff: remove nroff, tbl, and eqn — constructs ..... deroff(1)
Uutry: try to — contact remote system with debugging on ..... uutry(1M)
ls: list — contents of directory ..... ls(1)
sigcontext: get signal — context ..... sigcontext(2SEQ)
set and/or get signal stack — context ..... sigstack(2SEQ)
csplit: — context split ..... csplit(1)
sd: SCSI disk — controller ..... sd(7)
tm: SCSI tape — controller ..... tm(7)
ts: SCSI tape — controller ..... ts(7)
wd: SSM SCSI disk — controller ..... wd(7)
zd: dual-channel disk — controller ..... zd(7)
disk formatter for dual-channel disk — controller ..... zdformat(8)
synchronize a co-process with the — controlling FMLI object ..... vsig(1F)
virtually hangup the specified — controlling terminal ..... fvhangup(2SEQ)
tty: — controlling terminal (gentty) interface ..... tty(7)
_tolower, toascii: translate/ — conv: toupper, tolower, _toupper, ... conv(3C)
units: — conversion program ..... units(1)
character classification and — conversion tables ..... chrtbl(1M)
terminfo description captoinfo: — convert a termcap description into a ..... captoinfo(1M)
dd: — convert and copy a file ..... dd(1M)
long integers l3tol, ltol3: — convert between 3-byte integers and ..... l3tol(3C)
base-64 ASCII string a64l, l64a: — convert between long integer and ..... a64l(3C)
dev_to_ttyname, ttyname_to_dev: — convert major/minor numbers ..... dev_to_ttyname(3C)
/cftime, ascftime, mktime, tzset: — convert date and time to string ..... ctime(3C)
ecvt, fcvt, gcvt: — convert floating-point number to string ..... ecvt(3C)
scanf, fscanf, sscanf: — convert formatted input ..... scanf(3S)
strtod, atof: — convert string to double-precision number ..... strtod(3C)

strtol, atol, atoi: convert string to integer ..... strtol(3C)
password file scheme pwunconv: converts from a two to a one ..... pwunconv(1M)
vsig: synchronize a co-process with the controlling FMLI object ..... vsig(1F)
UNIX-to-UNIX system copy ..... uucp(1C)
system file copy ..... uuto(1C)
dd: convert and copy a file ..... dd(1M)
cpset: copy file and set mode, user, and group IDs ..... cpset(1)
cpio: copy file archives in and out ..... cpio(1)
cp, ln, mv: copy, link or move files ..... cp(1)
volcopy: make literal copy of filesystem ..... volcopy(1M)
transfer copy: perform a standalone data ..... copy(8)
savecore: save a core dump of the operating system ..... savecore(1M)
core: format of core image file ..... core(4)
core: format of core image file ..... core(4)
mem, kmem: core memory ..... mem(7)
and unset environment variables in core or in files ..... set(1F)
cocreate, cosend, cocheck, coreceive, codestroy: communicate to a/ ..... coproc(1F)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
trigonometric functions trig: sin, cos, tan, asin, acos, atan, atan2: ..... trig(3M)
codestroy: communicate to/ cocreate, cosend, cocheck, coreceive, ... coproc(1F)
sinh, cosh, tanh: hyperbolic functions ..... sinh(3M)
wc: word count ..... wc(1)
sum: print checksum and block count of a file ..... sum(1)
m_next: increment global counter ..... m_next(3PPS)
usclk: 32-bit microsecond counter ..... usclk(7)
cp, ln, mv: copy, link or move files ..... cp(1)
cpio: format of cpio archive ..... cpio(4)
cpio: copy file archives in and out ..... cpio(1)
cpio: format of cpio archive ..... cpio(4)
cpp: C preprocessor ..... see *C and Lang Tools Binder*
cprs: compress object file ..... see *C and Lang Tools Binder*
and group IDs cpset: copy file and set mode, user, ... cpset(1)
clock: report CPU time used ..... clock(3C)
cpus_online: returns the number of CPUs on-line ..... cpus_online(3PPS)
CPUs on-line cpus_online: returns the number of ..... cpus_online(3PPS)
crash: examine system images ..... crash(1M)
an existing one creat: create a new file or rewrite ..... creat(2)
setcolor: redefine or create a color ..... setcolor(1F)
gencc: create a front-end to the cc command ..... gencc(1M)
tmpnam, tempnam: create a name for a temporary file ..... tmpnam(3S)
creat: create a new file or rewrite an existing one ..... creat(2)
fork: create a new process ..... fork(2)
ctags: create a tags file ..... ctags(1)
tmpfile: create a temporary file ..... tmpfile(3S)
pipe: create an interprocess channel ..... pipe(2)
admin: create and administer SCCS files ..... admin(1)
tasktbl: create security table in shared memory ..... tasktbl(1M)
setsid: create session and set process group ID ..... setsid(2)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
cron: clock daemon ..... cron(1M)
cronlog: cron history file ..... cronlog(4)
cronlog: cron history file ..... cronlog(4)

crontab: user — crontab file ..... crontab(1)
crontab: user crontab file ..... crontab(1)
crontabs: clock daemon file ..... crontabs(4)
cxref: generate C program — cross-reference ..... see *C and Lang Tools Binder*
pg: file perusal filter for — CRTs ..... pg(1)
crypt: encode/decode ..... crypt(1)
functions — crypt: password and file encryption ..... crypt(3X)
hashing encryption — crypt, setkey, encrypt: generate ..... crypt(3C)
with C-like syntax — csh: a shell (command interpreter) ..... csh(1)
csplit: context split ..... csplit(1)
ct: spawn getty to a remote terminal ..... ct(1C)
ctags: create a tags file ..... ctags(1)
terminal — ctermid: generate filename for ..... ctermid(3S)
cftime, ascftime, mktime, tzset:/ — ctime, localtime, gmtime, asctime, ... ctime(3C)
ctrace: C prog debugger ..... see *C and Lang Tools Binder*
cu: call another UNIX system ..... cu(1C)
scr_dump: format of — curses screen image file ..... scr_dump(4)
optimization package — curses: terminal screen handling and ..... curses(3X)
the user — cuserid: get character login name of ..... cuserid(3S)
line of a file — cut: cut out selected fields of each ..... cut(1)
cut: — cut out selected fields of each line of a file ..... cut(1)
cross-reference — cxref: generate C program ..... see *C and Lang Tools Binder*
backup.codes: backup — cycle code definitions ..... backup_codes(4)
bdflush: update — daemon ..... bdflush(1M)
bdflush: update — daemon ..... bdflush(2SEQ)
cron: clock — daemon ..... cron(1M)
strerr: STREAMS error logger — daemon ..... strerr(1M)
crontabs: clock — daemon file ..... crontabs(4)
creation pwdhd: — daemon to oversee password database ..... pwdhd(1M)
runacct: run — daily accounting ..... runacct(1M)
300s: handle special functions of — DASI 300 and 300s terminals ..... 300(1)
450: handle special functions of the — DASI 450 terminal ..... 450(1)
zcat: compress and expand — data ..... compress(1)
prof: display profile — data ..... prof(1)
time a command; report process — data and system activity ..... timex(1)
termcap: terminal capability — data base ..... termcap(4)
diskusg: generate disk accounting — data by user ID ..... diskusg(1M)
t_rcvuderr: receive a unit — data error indication ..... t_rcvuderr(3N)
sputl, sgetl: access long integer — data in a machine-independent/ ..... sputl(3X)
plock: lock process, text, or — data in memory ..... plock(2)
t_snd: send — data or expedited data over a connection ..... t_snd(3N)
t_rcv: receive — data or expedited data sent over a connection ..... t_rcv(3N)
t_snd: send data or expedited — data over a connection ..... t_snd(3N)
stat: — data returned by stat system call ..... stat(5)
brk, sbrk: change private — data segment size ..... brk(3PPS)
shbrk, shsbrk: change shared — data segment size ..... shbrk(3PPS)
brk, sbrk: change — data segment space allocation ..... brk(2)
t_rcv: receive data or expedited — data sent over a connection ..... t_rcv(3N)
copy: perform a standalone — data transfer ..... copy(8)
types: primitive system — data types ..... types(5)
t_rcvudata: receive a — data unit ..... t_rcvudata(3N)
t_sndudata: send a — data unit ..... t_sndudata(3N)

diskinfo: disk device characteristic database....diskinfo(4)
helpadm: change the Help Facility database....helpadm(1M)
terminfo: terminal capability database....terminfo(4)
a terminal or query terminfo database....tput(1)
pwdhd: daemon to oversee password database creation....pwdhd(1M)
mkpwdbm: make a database for fast password lookup....mkpwdbm(1M)
join: relational database operator....join(1)
dbm_clearerr, dmb_blkflush: database subroutines....ndbm(3X)
makewhatis: generate man page databases....makealiases(1M)
date: print and set the date....date(1)
ascftime, mktime, tzset: convert date and time to string....ctime(3C)
date: print and set the date....date(1)
/dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush: database/....ndbm(3X)
dbm_delete,/ dbm_open, dbm_close,....ndbm(3X)
dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey,/....ndbm(3X)
/dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr,/....ndbm(3X)
dbm_firstkey,/dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete,....ndbm(3X)
/dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error,/....ndbm(3X)
dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error,/....ndbm(3X)
dbm_store, dbm_delete,/ dbm_open, dbm_close, dbm_fetch,....ndbm(3X)
dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey,/....ndbm(3X)
pdbx, dbx: parallel debugger....see *C and Lang Tools Binder*
dc: desk calculator....dc(1)
enable/disable patrol seeks on DCC disks....sweepon(1M)
consistency check dcheck: filesystem directory....dcheck(1M)
dd: convert and copy a file....dd(1M)
ctrace: C program debugger....see *C and Lang Tools Binder*
pdbx, dbx: parallel debugger....see *C and Lang Tools Binder*
try to contact remote system with debugging on....uutry(1M)
crypt: encode/decode....crypt(1)
timezone: set default system time zone....timezone(4)
reset the current form field to its default values....reset(1F)
addusr.def: add user defaults....addusr_def(4)
format disks or perform other defect management functions....format(1M)
gtty: set and get terminal state (defunct)....stty(2)
ut_add_user, ut_delete_user: add/ delete users in utmp file....utadduser(3X)
the delta commentary of an SCCS delta....cdc(1)
delta: make a delta (change) to an SCCS file....delta(1)
cdc: change the delta commentary of an SCCS delta....cdc(1)
rmdel: remove a delta from an SCCS file....rmdel(1)
SCCS file delta: make a delta (change) to an....delta(1)
comb: combine SCCS deltas....comb(1)
constructs deroff: remove nroff, tbl, and eqn....deroff(1)
whatis: describe what a command is....whatis(1)
close: close a file descriptor....close(2)
dup: duplicate an open file descriptor....dup(2)
dup2: duplicate an open file descriptor....dup2(2)
setmntent: get file system descriptor file entry....getmntent(3SEQ)
getdtablemax: get maximum in-use descriptor table index....getdtablemax(2SEQ)
getdtablesize: get descriptor table size....getdtablesize(2SEQ)
setdtablesize: set descriptor table size....setdtablesize(2SEQ)
dc: desk calculator....dc(1)

devbuild, devdestroy: build and destroy virtual devices ..... devbuild(1M)
devbuild, devdestroy: build and ..... devbuild(1M)
virtual devices devbuild, devdestroy: build and destroy ..... devbuild(1M)
ioctl: control device ..... ioctl(2)
prtvtoc: print the VTOC of a block device ..... prtvtoc(1M)
print the VTOC of a standalone disk device ..... prtvtoc(8)
sp: STREAMS pipe device ..... sp(7)
or the terminal associated with a device ..... ttydevname(1)
the terminal/ ttydevname: get the device associated with a terminal or ..... ttydevname(1)
diskinfo: disk device characteristic database ..... diskinfo(4)
devnm: device name ..... devnm(1M)
clone: open any minor device on a STREAMS driver ..... clone(7)
devtab: virtual device table ..... devtab(4)
dump kernel auto-configuration device table ..... dumpconf(1M)
backup.dev: backup devices ..... backup_dev(4)
build and destroy virtual devices ..... devbuild(1M)
of virtual-to-physical mapping devices ..... pmap(1M)
SA: devices administered by System Administration ..... sa(7)
constab: constructed devices table ..... constab(4)
devnm: device name ..... devnm(1M)
devtab: virtual device table ..... devtab(4)
convert between major/minor numbers/ dev_to_ttyname, ttyname_to_dev: ..... dev_to_ttyname(3C)
blocks and inodes df: report number of free disk ..... df(1M)
dmesg: collect system diagnostic messages to form error log ..... dmesg(1M)
SCED: SCSI/Ethernet/ Diagnostics board ..... sced(7)
terminal line connection dial: establish an out-going ..... dial(3C)
bdiff: big diff ..... bdiff(1)
diff: differential file comparison ..... diff(1)
comparison diff3: 3-way differential file ..... diff3(1)
sdiff: side-by-side difference program ..... sdiff(1)
diff: differential file comparison ..... diff(1)
diff3: 3-way differential file comparison ..... diff3(1)
dir: format of directories ..... dir(4)
dircmp: directory comparison ..... dircmp(1)
dir: format of directories ..... dir(4)
unlink: link and unlink files and directories ..... link(1M)
mkdir: make directories ..... mkdir(1)
rm, rmdir: remove files or directories ..... rm(1)
uucheck: check the uucp directories and permissions file ..... uucheck(1M)
cd: change working directory ..... cd(1)
chdir: change working directory ..... chdir(2)
chroot: change root directory ..... chroot(2)
get pathname of current working directory ..... getcwd(3C)
system level transition files directory ..... init.d(4)
ls: list contents of directory ..... ls(1)
mkdir: make a directory ..... mkdir(2)
mvdir: move a directory ..... mvdir(1M)
rmdir: remove a directory ..... rmdir(2)
uucleanup: uucp spool directory clean-up ..... uucleanup(1M)
dircmp: directory comparison ..... dircmp(1)
dcheck: filesystem directory consistency check ..... dcheck(1M)
system independent/ getdents: read directory entries and put in a file ..... getdents(2)

dirent: file system independent — directory entry ..... dirent(4)
unlink: remove — directory entry ..... unlink(2)
chroot: change root — directory for a command ..... chroot(1M)
pwd: working — directory name ..... pwd(1)
telldir, seekdir, rewinddir,/ — directory: opendir, readdir, ... directory(3X)
seekdir, rewinddir, closedir: — directory operations ..... directory(3X)
mknod: make a — directory, or a special or ordinary file ..... mknod(2)
directory entry — dirent: file system independent ..... dirent(4)
basename, — dirname: deliver portions of pathnames ..... basename(1)
dis: code disassembler ..... see *C and Lang Tools Binder*
instructions — dis386: disassemble Series 386 ..... dis386(3SEQ)
t_unbind: — disable a transport endpoint ..... t_unbind(3N)
enable, — disable: enable/disable LP printers ..... enable(1)
enable, disable: enable/ — disable LP printers ..... enable(1)
sweepon, sweepoff: enable/ — disable patrol seeks on DCC disks ..... sweepon(1M)
acct: enable or — disable process accounting ..... acct(2)
swapping, or processor migration — disabled ..... noage(1M)
dis386: — disassemble Series 386 instructions ..... dis386(3SEQ)
dis: object code — disassembler ..... see *C and Lang Tools Binder*
t_rcvdis: retrieve information from — disconnect ..... t_rcvdis(3N)
t_snddis: send user-initiated — disconnect request ..... t_snddis(3N)
installs a boot program on a — disk ..... installboot(1M)
diskusg: generate — disk accounting data by user ID ..... diskusg(1M)
df: report number of free — disk blocks and inodes ..... df(1M)
sd: SCSI — disk controller ..... sd(7)
wd: SSM SCSI — disk controller ..... wd(7)
zd: dual-channel — disk controller ..... zd(7)
disk formatter for dual-channel — disk controller ..... zdformat(8)
print the VTOC of a standalone — disk device ..... prtvtoc(8)
diskinfo: — disk device characteristic database ..... diskinfo(4)
CCSformat: standalone embedded SCSI — disk formatter ..... ccsformat(8)
controller zdformat: standalone — disk formatter for dual-channel disk ..... zdformat(8)
getgeombyname: get — disk geometry by name ..... getgeombyname(3SEQ)
quotactl: manipulate — disk quotas ..... quotactl(2SEQ)
du: summarize — disk usage ..... du(1M)
quota: display — disk usage and limits ..... quota(1)
database — diskinfo: disk device characteristic ..... diskinfo(4)
mkvtoc: populate VTOC on — disks ..... mkvtoc(1M)
enable/disable patrol seeks on DCC — disks ..... sweepon(1M)
mgmt functions format: format — disks or perform other defect ..... format(1M)
data by user ID — diskusg: generate disk accounting ..... diskusg(1M)
and/or the working/ indicator: — display application specific alarms ..... indicator(1F)
quota: — display disk usage and limits ..... quota(1)
vi: screen-oriented (visual) — display editor based on ex ..... vi(1)
man: — display on-line man pages ..... man(1)
virtual-to-physical mapping/ pmap: — display or alter parameters of ..... pmap(1M)
prof: — display profile data ..... prof(1)
bootflags: — display/change system boot flags ..... bootflags(1M)
hypot: Euclidean — distance function ..... hypot(3M)
dltsectbl: refresh security table ..... dltsectbl(1M)
/dbm_error, dbm_clearerr, — dmb_blkflush: database subroutines ..... ndbm(3X)
messages to form error log — dmesg: collect system diagnostic ..... dmesg(1M)

checkeq: check documents for correct eqn syntax.....checkeq(1)
prctmp,/ chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm,.....acctsh(1M)
strtod, atof: convert string to double-precision number.....strtod(3C)
mrand48, jrand48, srand48, seed48,/ drand48, erand48, lrand48, nrand48,.....drand48(3C)
open any minor device on a STREAMS driver.....clone(7)
cmpt: compatibility partitioning driver.....cmpt(7)
cn: pseudoconsole driver.....cn(7)
terminal multiplexor driver.....st(7)
streampty: STREAMS pseudo terminal driver.....streampty(7)
vtoc: VTOC partitioning driver.....vtoc(7)
Xylogics 781 terminal multiplexor driver.....xs(7)
streamtty: Generic STREAMS tty driver interface.....streamtty(7)
consmem, sm, ss: console memory drivers.....consmem(7)
du: summarize disk usage.....du(1M)
zd: dual-channel disk controller.....zd(7)
/standalone disk formatter for dual-channel disk controller.....zdformat(8)
dump: incremental filesystem dump.....dump(1M)
dump: standalone memory dump.....dump(8)
od: octal dump.....od(1)
of an object file dump: dump parts.....see *C and Lang Tools Binder*
format dump, dumpdates: incremental dump.....dump(4)
dumpfs: dump filesystem information.....dumpfs(1M)
dump, dumpdates: incremental dump format.....dump(4)
dump: incremental filesystem dump.....dump(1M)
device table dumpconf: dump kernel auto-configuration.....dumpconf(1M)
savecore: save a core dump of the operating system.....savecore(1M)
dump: standalone memory dump.....dump(8)
auto-configuration device table dumpconf: dump kernel.....dumpconf(1M)
dump, dumpdates: incremental dump format.....dump(4)
dumpfs: dump filesystem information.....dumpfs(1M)
descriptor dup: duplicate an open file.....dup(2)
descriptor dup2: duplicate an open file.....dup2(2)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
ec: forms compiler.....ec(1M)
echo: echo arguments.....echo(1)
echo: echo arguments.....echo(1)
echo: put string on virtual output.....echo(1F)
floating-point number to string ecvt, fcvt, gcvt: convert.....ecvt(3C)
ed, red: text editor.....ed(1)
end, etext, edata: last locations in program.....end(3C)
command edc: run an externally described.....edc(1M)
casual users) edit: text editor (variant of ex for.....edit(1)
edquota: edit user quotas.....edquota(1M)
sact: print current SCCS file editing activity.....sact(1)
ed, red: text editor.....ed(1)
ex: text editor.....ex(1)
sed: stream editor.....sed(1)
vi: screen-oriented (visual) display editor based on ex.....vi(1)
ld: link editor for object files.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
edit: text editor (variant of ex for casual users).....edit(1)

edquota: edit user quotas ..... edquota(1M)
effective group IDs ..... getuid(2)
effective user, real group, and ..... getuid(2)
efl: Extended Fortran Language ..... efl(1)
efl files ..... fsplit(1)
egrep: search a file for a pattern ..... egrep(1)
embedded SCSI disk formatter ..... ccsformat(8)
enable, disable: enable/disable LP ..... enable(1)
enable or disable process accounting ..... acct(2)
enable/disable LP printers ..... enable(1)
enable/disable patrol seeks on DCC disks ..... sweepon(1M)
encode/decode ..... crypt(1)
encrypt: generate hashing encryption ..... crypt(3C)
encryption ..... crypt(3C)
encryption functions ..... crypt(3X)
encryption key ..... makekey(1)
end, etext, edata: last locations in ..... end(3C)
end to the cc command ..... gencc(1M)
endgrent, fgetgrent: get group file entry ..... getgrent(3C)
endmntent, getmntent, hasmntopt, ... getmntent(3SEQ)
endpoint ..... t_bind(3N)
endpoint ..... t_close(3N)
endpoint ..... t_look(3N)
endpoint ..... t_open(3N)
endpoint ..... t_optmgmt(3N)
endpoint ..... t_unbind(3N)
endpwent, fgetpwent: get password file/ ..... getpwent(3C)
endspent, fgetspent, lckpwdf/ ..... getspent(3X)
endutent, utmpname, getutpid: access/ ..... getut(3C)
enhancement request ..... mailbug(1)
env: set environment for command ..... env(1)
.environ, .pref, .variables: FACE ..... env(4)
environ: user environment ..... environ(5)
environment ..... environ(5)
environment ..... putenv(3C)
environment ..... rc2(1M)
environment at login time ..... profile(4)
environment control ..... fpgetround(3C)
environment for command execution ..... env(1)
environment name ..... getenv(3C)
environment variables in core or in files ..... set(1F)
eqn constructs ..... deroff(1)
eqn syntax ..... checkeq(1)
erand48, lrand48, nrand48, mrand48, ... drand48(3C)
erf, erfc: error function and ..... erf(3M)
erfc: error function and ..... erf(3M)
errno, sys_errlist, sys_nerr: system ..... perror(3C)
error function ..... erf(3M)
error function and complementary ..... erf(3M)
error indication ..... t_rcvuderr(3N)
error log ..... dmesg(1M)
error log file ..... errorlog(4)

effective user, real group, and
/getgid, getegid: get real user,

fsplit: split f77, ratfor, or
using full regular expressions

CCSformat: standalone
printers

acct:

enable, disable:

sweepon, sweepoff:

crypt:

crypt, setkey,
setkey, encrypt: generate hashing

crypt: password and file
makekey: generate
program

gencc: create a front-
/getgrgid, getgrnam, setgrent,
setmntent: get file/ addmntent,

bind an address to a transport

t_close: close a transport
at the current event on a transport

t_open: establish a transport
manage options for a transport

t_unbind: disable a transport
/getpwuid, getpwnam, setpwent,
getspent, getspnam, setspent,
/getutline, pututline, setutent,

mailbug: submit a bug report or
execution
files

environ: user

putenv: change or add value to
commands performed for multi-user

profile: setting up an

fpsetsticky: IEEE floating-point

env: set

getenv: return value for

set, unset: set and unset

deroff: remove nroff, tbl, and
checkeq: check documents for correct

jrand48, srand48, seed48,/ drand48,
complementary error function

complementary error function erf,
error messages perror,

error function and complementary
error function erf, erfc:

t_rcvuderr: receive a unit data
system diagnostic messages to form

errorlog: internal

strclean: STREAMS error logger cleanup program ..... strclean(1M)
strerr: STREAMS error logger daemon ..... strerr(1M)
log: interface to STREAMS error logging and event tracing ..... log(7)
t_error: produce error message ..... t_error(3N)
errormsg: error message text file ..... errormsg(4)
errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
introduction to system calls and error numbers ..... intro(2)
matherr: error-handling function ..... matherr(3M)
errorlog: internal error log file ..... errorlog(4)
errormsg: error message text file ..... errormsg(4)
spellin, hashcheck: find spelling errors ..... spell(1)
/etc/shadow with information from / etc/passwd ..... pwconv(1M)
pwconv: installs and updates / etc/shadow with information from/ ..... pwconv(1M)
end, etext, edata: last locations in program ..... end(3C)
SCED: SCSI/ Ethernet/Diagnostics board ..... sced(7)
hypot: Euclidean distance function ..... hypot(3M)
expr: evaluate arguments as an expression ..... expr(1)
test: condition evaluation command ..... test(1)
t_look: look at the current event on a transport endpoint ..... t_look(3N)
to STREAMS error logging and event tracing ..... log(7)
sulog: log of substitute user events ..... sulog(4)
(visual) display editor based on ex ..... vi(1)
edit: text editor (variant of ex for casual users) ..... edit(1)
ex: text editor ..... ex(1)
execlp, execvp: execute a file exec: execl, execv, execle, execve, ... exec(2)
execlp, execvp: execute a/ exec: execl, execv, execle, execve, ... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
run: run an executable ..... run(1F)
execle, execve, execlp, execvp: execute a file ..... exec(2)
m_fork: execute a subprogram in parallel ..... m_fork(3PPS)
construct argument list(s) and execute command ..... xargs(1)
at, batch: execute commands at a later time ..... at(1)
regcmp, regex: compile and execute regular expression ..... regcmp(3X)
uuxqt: execute remote command requests ..... uuxqt(1M)
env: set environment for command execution ..... env(1)
suspend and resume child process execution ..... m_park_procs(3PPS)
/system command execution ..... uux(1C)
sleep: suspend execution for an interval ..... sleep(1)
sleep: suspend execution for interval ..... sleep(3C)
monitor: prepare execution profile ..... monitor(3C)
profil: execution time profile ..... profil(2)
execvp: execute a file exec: execl, execv, execle, execlp, ... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
exit, _exit: terminate process ..... exit(2)
_exit: terminate process ..... exit(2)
exponential, logarithm, power/ exp, log, log10, pow, sqrt: ..... exp(3M)
uncompress, zcat: compress and expand data ..... compress(1)
pack, pcat, unpack: compress and expand files ..... pack(1)
exp, log, log10, pow, sqrt: exponential, logarithm, power, square/ ..... exp(3M)
expression expr: evaluate arguments as an ..... expr(1)

expr: evaluate arguments as an expression ..... expr(1)
regex: compile and execute regular expression ..... regcmp(3X)
regcmp: regular expression compile ..... regcmp(1)
regexp: regular expression compile and match routines ..... regexp(5)
for a pattern using full regular expressions ..... egrep(1)
efl: Extended Fortran Language ..... efl(1)
edc: run an externally described command ..... edc(1M)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ ..... floor(3M)
pathalias: alias file for FACE ..... alias(4)
.environ, .pref, .variables: FACE files ..... env(4)
report inter-process communication facilities status ..... ipcs(1)
a number factor: obtain the prime factors of ..... factor(1)
factor: obtain the prime factors of a number ..... factor(1)
true, false: provide truth values ..... true(1)
abort: generate an IOT fault ..... abort(3C)
stream fclose, fflush: close or flush a ..... fclose(3S)
fcntl: file control ..... fcntl(2)
fcntl: file control options ..... fcntl(5)
number to string ecvt, fcvt, gcvt: convert floating-point ..... ecvt(3C)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
stream status inquiries ferror, feof, clearerr, fileno: ..... ferror(3S)
for a filesystem ff: list filenames and statistics ..... ff(1M)
fclose, fflush: close or flush a stream ..... fclose(3S)
from a stream getc, getchar, fgetc, getw: get character or word ..... getc(3S)
/getgrnam, setgrent, endgrent, fgetgrent: get group file entry ..... getgrent(3C)
/getpwnam, setpwent, endpwent, fgetpwent: get password file entry ..... getpwent(3C)
gets, fgets: get a string from a stream ..... gets(3S)
/getspnam, setspent, endspent, fgetspent, lckpwdf, ulckpwdf: get/ ..... getspent(3X)
string fgrep: search a file for a character ..... fgrep(1)
cut: cut out selected fields of each line of a file ..... cut(1)
mkfifo: make a FIFO special file ..... mkfifo(3C)
access: determine accessibility of a file ..... access(2)
chmod: change mode of file ..... chmod(2)
chown: change owner and group of a file ..... chown(2)
core: format of core image file ..... core(4)
cronlog: cron history file ..... cronlog(4)
crontab: user crontab file ..... crontab(1)
crontabs: clock daemon file ..... crontabs(4)
ctags: create a tags file ..... ctags(1)
selected fields of each line of a file ..... cut(1)
dd: convert and copy a file ..... dd(1M)
make a delta (change) to an SCCS file ..... delta(1)
errorlog: internal error log file ..... errorlog(4)
errormsg: error message text file ..... errormsg(4)
execve, execlp, execvp: execute a file ..... exec(2)
get: get a version of an SCCS file ..... get(1)
group: group file ..... group(4)
issue: issue identification file ..... issue(4)
header of a member of an archive file ..... ldahread(3X)
ldaclose: close a common object file ..... ldclose(3X)

the file header of a common object file.....ldfhread(3X)
of a section of a common object file.....ldlseek(3X)
file header of a common object file.....ldohseek(3X)
of a section of a common object file.....ldrseek(3X)
section header of a common object file.....ldshread(3X)
section of a common object file.....ldsseek(3X)
table entry of a common object file.....ldtbindex(3X)
table entry of a common object file.....ldtbread(3X)
the symbol table of a common object file.....ldtbseek(3X)
number entries in a common object file.....linenum(4)
link: link to a file.....link(2)
mkfifo: make a FIFO special file.....mkfifo(3C)
mknod: build special file.....mknod(1M)
directory, or a special or ordinary file.....mknod(2)
format of a module configuration file.....mod_config(4)
motd: message-of-the-day file.....motd(4)
newform: change the format of a text file.....newform(1)
null: the null file.....null(7)
passwd: password file.....passwd(4)
files or subsequent lines of one file.....paste(1)
product: installed products file.....product(4)
prs: print an SCCS file.....prs(1)
rcsfile: format of RCS file.....rcsfile(4)
read: read from file.....read(2)
readv: read vector from file.....readv(2SEQ)
reinit: run an initialization file.....reinit(1F)
information for a common object file.....reloc(4)
rename: change the name of a file.....rename(2)
rmdel: remove a delta from an SCCS file.....rmdel(1)
compare two versions of an SCCS file.....sccsdiff(1)
sccsfile: format of SCCS file.....sccsfile(4)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
section header for a common object file.....scnhdr(4)
format of curses screen image file.....scr_dump(4)
shadow: restricted password file.....shadow(4)
system shutdown information file.....shutdown(4)
spellhist: unmatched spelling words file.....spellhist(4)
print checksum and block count of a file.....sum(1)
symlink: make symbolic link to a file.....symlink(2SEQ)
format of a system configuration file.....sys_config(4)
tail: deliver the last part of a file.....tail(1)
term: format of compiled term file.....term(4)
tmpfile: create a temporary file.....tmpfile(3S)
create a name for a temporary file.....tmpnam(3S)
access and modification times of a file.....touch(1)
undo a previous get of an SCCS file.....unget(1)
uniq: report repeated lines in a file.....uniq(1)
add/delete users in utmp file.....utadduser(3X)
the uucp directories and permissions file.....uucheck(1M)
val: validate SCCS file.....val(1)
write: write on a file.....write(2)
writev: write vector on a file.....writev(2SEQ)

utime: set file access and modification times.....utime(2)
ldfcn: common object file access routines.....ldfcn(4)
readfile, longline: reads file and gets longest line.....readfile(1F)
cpset: copy file and set mode, user, and group IDs.....cpset(1)
Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
tar: tape file archiver.....tar(1)
cpio: copy file archives in and out.....cpio(1)
rcs: change RCS file attributes.....rcs(1)
pwck, grpck: password/group file checkers.....pwck(1M)
mcs: manipulate the object file comment section.....mcs(1)
diff: differential file comparison.....diff(1)
diff3: 3-way differential file comparison.....diff3(1)
fcntl: file control.....fcntl(2)
fcntl: file control options.....fcntl(5)
UNIX-to-UNIX system file copy.....uuto(1C)
umask: set and get file creation mask.....umask(2)
close: close a file descriptor.....close(2)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
file: determine file type.....file(1)
sact: print current SCCS file editing activity.....sact(1)
crypt: password and file encryption functions.....crypt(3X)
endgrent, fgetgrent: get group file entry.....getgrent(3C)
get file system descriptor file entry.....getmntent(3SEQ)
endpwent, fgetpwent: get password file entry.....getpwent(3C)
ulckpwdf: get shadow password file entry.....getspent(3X)
utmpname, getutpid: access utmp file entry.....getut(3C)
putpwent: write password file entry.....putpwent(3C)
putspent: write shadow password file entry.....putspent(3X)
fgrep: search a file for a character string.....fgrep(1)
grep: search a file for a pattern.....grep(1)
egrep: search a file for a pattern using full regular expressions.....egrep(1)
proto: prototype job file for at.....proto(4)
pathalias: alias file for FACE.....alias(4)
ldaopen: open a common object file for reading.....ldopen(3X)
acct: per-process accounting file format.....acct(4)
ar: common archive file format.....see *C and Lang Tools Binder*
manager, updates text files in ISIS file format.....isismgr(1M)
intro: introduction to file formats.....intro(4)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
number entries of a common object file function.....ldlread(3X)
filehdr: file header for common object files.....filehdr(4)
limits: file header for implementation-specific/.....limits(4)
unistd: file header for symbolic constants.....unistd(4)
ldfhread: read the file header of a common object file.....ldfhread(3X)
ldohseek: seek to the optional file header of a common object file.....ldohseek(3X)
split: split a file into pieces.....split(1)
mmap: map an open file into the processs address space.....mmap(2SEQ)
ISIS file format isismgr: ISIS file manager, updates text files in.....isismgr(1M)
merge: three-way file merge.....merge(1)
ttyslot: find the slot in the utmp file of the current user.....ttyslot(3C)
more: view a file one full screen at a time.....more(1)

fuser: identify processes using a — file or file structure ..... fuser(1M)
creat: create a new — file or rewrite an existing one ..... creat(2)
pg: — file perusal filter for CRTs ..... pg(1)
lseek: move read/write — file pointer ..... lseek(2)
fseek, rewind, ftell: reposition a — file pointer in a stream ..... fseek(3S)
bfs: big — file scanner ..... bfs(1)
from a two to a one password — file scheme ..... pwunconv(1M)
stat, lstat, fstat: get — file status ..... stat(2)
identify processes using a file or — file structure ..... fuser(1M)
symbol name for common object — file symbol table entry ..... ldgetname(3X)
syms: common object — file symbol table format ..... syms(4)
newfs: construct a new — file system ..... newfs(1M)
/read directory entries and put in a — file system independent format ..... getdents(2)
/getmntent, hasmntopt, setmntent: get — file system descriptor file entry ..... getmntent(3SEQ)
dirent: — file system independent directory entry ..... dirent(4)
fs, inode: format of — file system volume ..... fs(4)
truncate, ftruncate: truncate a — file to a specified length ..... truncate(2SEQ)
uusched: the scheduler for the uucp — file transport program ..... uusched(1M)
uucico: — file transport program for the uucp system ..... uucico(1M)
ftw: walk a — file tree ..... ftw(3C)
file: determine — file type ..... file(1)
umask: set — file-creation mode mask ..... umask(1)
object files — filehdr: file header for common ..... filehdr(4)
mktemp: make a unique — filename ..... mktemp(3C)
search Interpreter criteria for — filename ..... pathconv(1F)
ctermid: generate — filename for terminal ..... ctermid(3S)
ff: list — filenames and statistics for a filesystem ..... ff(1M)
ferror, feof, clearerr, — fileno: stream status inquiries ..... ferror(3S)
search and print process accounting — files ..... acctcom(1)
merge or add total accounting — files ..... acctmerg(1M)
admin: create and administer SCCS — files ..... admin(1)
cat: concatenate and print — files ..... cat(1)
cmp: compare two — files ..... cmp(1)
or reject lines common to two sorted — files ..... comm(1)
config: build system configuration — files ..... config(1M)
cp, ln, mv: copy, link or move — files ..... cp(1)
.environ, .pref, .variables: FACE — files ..... env(4)
file header for common object — files ..... filehdr(4)
find: find — files ..... find(1)
fspec: format specification in text — files ..... fspec(4)
fsplit: split f77, ratfor, or efl — files ..... fsplit(1)
string, format of graphical — files ..... gps(4)
ident: identify — files ..... ident(1)
intro: introduction to special — files ..... intro(7)
lockf: record locking on — files ..... lockf(3C)
makedev: make system special — files ..... makedev(1M)
pcat, unpack: compress and expand — files ..... pack(1)
pr: print — files ..... pr(1)
and other information about RCS — files ..... rlog(1)
environment variables in core or in — files ..... set(1F)
sort: sort and/or merge — files ..... sort(1)
uucp: networking support — files ..... uucp(4)

what: identify SCCS files ..... what(1)
link, unlink: link and unlink files and directories ..... link(1M)
init.d: system level transition files directory ..... init.d(4)
ISIS file manager, updates text files in ISIS file format ..... isismgr(1M)
passmgmt: password files management ..... passmgmt(1M)
rm, rmdir: remove files or directories ..... rm(1)
paste: merge same lines of several files or subsequent lines of one/ ..... paste(1)
ott: files that hold object architecture information ..... ott(4)
checkfsys: check a filesystem ..... checkfsys(1M)
list filenames and statistics for a filesystem ..... ff(1M)
mkfs: construct a filesystem ..... mkfs(1M)
mount: mount a filesystem ..... mount(2)
repquota: summarize quotas for a filesystem ..... repquota(1M)
tunefs: tune up an existing filesystem ..... tunefs(1M)
umount: unmount a filesystem ..... umount(2)
volcopy: make literal copy of filesystem ..... volcopy(1M)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
interactive repair fsck: filesystem consistency check and ..... fsck(1M)
dcheck: filesystem directory consistency check ..... dcheck(1M)
dump: incremental filesystem dump ..... dump(1M)
fstyp: determine filesystem identifier ..... fstyp(1M)
dumpfs: dump filesystem information ..... dumpfs(1M)
statfs, fstatfs: get filesystem information ..... statfs(2)
quotacheck: filesystem quota consistency checker ..... quotacheck(1M)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
restore: incremental filesystem restore ..... restore(1M)
ustat: get filesystem statistics ..... ustat(2)
fsstat: report filesystem status ..... fsstat(1M)
icheck: filesystem storage consistency check ..... icheck(1M)
mnttab: mounted filesystem table ..... mnttab(4)
labelit: provide labels for filesystems ..... labelit(1M)
mount, umount: mount and unmount filesystems ..... mount(1M)
umountall: mount, unmount multiple filesystems ..... mountall(1M)
checklist: list of filesystems processed by fsck and ncheck ..... checklist(4)
fstab: file-system-table ..... fstab(4)
greek: select terminal filter ..... greek(1)
nl: line numbering filter ..... nl(1)
pg: file perusal filter for CRTs ..... pg(1)
col: filter reverse line-feeds ..... col(1)
lpfilter: administer filters used with the LP print service ..... lpfilter(1M)
find: find files ..... find(1)
for an object library lorder: find ordering relation ..... see *C and Lang Tools Binder*
display/change system boot flags ..... bootflags(1M)
/fpgetsticky, fpsetsticky: IEEE floating-point environment control ..... fpgetround(3C)
isnan: isnand, isnanf: test for floating-point NaN (Not-A-Number) ..... isnan(3C)
ecvt, fcvt, gcvt: convert floating-point number to string ..... ecvt(3C)
ldexp, modf: manipulate parts of floating-point numbers ..... frexp(3C)
ceiling, remainder, absolute value/ floor, ceil, fmod, fabs: floor, ... floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ ..... floor(3M)
fclose, fflush: close or flush a stream ..... fclose(3S)
a co-process with the controlling FMLI object ..... vsig(1F)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute/ ..... floor(3M)

stream fopen, freopen, fdopen: open a ..... fopen(3S)
tcgetpgrp: get foreground process group ID..... tcgetpgrp(2)
tcsetpgrp: set foreground process group ID..... tcsetpgrp(2)
fork: create a new process..... fork(2)
system diagnostic messages to form error log..... dmesg(1M)
reset: reset the current form field to its default values ..... reset(1F)
acct: per-process accounting file format ..... acct(4)
dump, dumpdates: incremental dump format..... dump(4)
put in a file system independent format..... getdents(2)
updates text files in ISIS file format..... isismgr(1M)
common object file symbol table format..... syms(4)
management functions format: format disks or perform other defect..... format(1M)
other defect management functions format: format disks or perform ..... format(1M)
mod_config: format of a module configuration file..... mod_config(4)
sys_config: format of a system configuration file..... sys_config(4)
newform: change the format of a text file..... newform(1)
term: format of compiled term file..... term(4)
core: format of core image file..... core(4)
cpio: format of cpio archive..... cpio(4)
scr_dump: format of curses screen image file..... scr_dump(4)
dir: format of directories..... dir(4)
fs, inode: format of file system volume ..... fs(4)
gps: graphical primitive string, format of graphical files..... gps(4)
rcsfile: format of RCS file..... rcsfile(4)
sccsfile: format of SCCS file ..... sccsfile(4)
fspec: format specification in text files..... fspec(4)
strftime: format time information ..... strftime(3C)
intro: introduction to file formats ..... intro(4)
utmp, wtmp: utmp and wtmp entry formats..... utmp(4)
scanf, fscanf, sscanf: convert formatted input ..... scanf(3S)
printf, fprintf, sprintf: print formatted output ..... printf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs/..... vprintf(3S)
pwfix: remove incorrectly formatted password entries..... pwfix(1M)
standalone embedded SCSI disk formatter..... ccsformat(8)
zdformat: standalone disk formatter for dual-channel disk/..... zdformat(8)
ASSIST menus or command forms..... astgen(1)
ec: forms compiler..... ec(1M)
lpforms: administer forms used with the LP print service..... lpforms(1M)
addformopt: add a form/SELECT window option to a menu ..... addformopt(1M)
ratfor: rational Fortran dialect..... ratfor(1)
efl: Extended Fortran Language ..... efl(1)
variables pathconf, fpathconf: get configurable pathname..... pathconf(2)
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky,/..... fpgetround(3C)
fpsetmask, fpgetsticky,/ fpgetround, fpsetround, fpgetmask, ... fpgetround(3C)
/fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky: IEEE/..... fpgetround(3C)
printf, fprintf, sprintf: print formatted output..... printf(3S)
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky:/ ..... fpgetround(3C)
fpgetsticky,/ fpgetround, fpsetround, fpgetmask, fpsetmask, ... fpgetround(3C)
/fpgetmask, fpsetmask, fpgetsticky, fpsetsticky: IEEE floating-point/ ..... fpgetround(3C)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
puts, fputs: put a string on a stream..... puts(3S)
getfrm: returns the current frame number ..... getfrm(1F)

fread, fwrite: binary input/output ..... fread(3S)
t_free: free a library structure ..... t_free(3N)
df: report number of free disk blocks and inodes ..... df(1M)
malloc, free, realloc, calloc: main memory allocator ..... malloc(3C)
mallinfo: fast main memory/ malloc, free, realloc, calloc, mallopt, ... malloc(3X)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
of floating-point numbers frexp, ldexp, modf: manipulate parts ..... frexp(3C)
gencc: create a front-end to the cc command ..... gencc(1M)
volume fs, inode: format of file system ..... fs(4)
scanf, fscanf, sscanf: convert formatted input ..... scanf(3S)
list of filesystems processed by fsck and ncheck ..... checklist(4)
and interactive repair fsck: filesystem consistency check ..... fsck(1M)
file pointer in a stream fseek, rewind, ftell: reposition a ..... fseek(3S)
generation numbers fsirand: install random inode ..... fsirand(1M)
files fspec: format specification in text ..... fspec(4)
files fsplit: split f77, ratfor, or efl ..... fsplit(1)
fsstat: report filesystem status ..... fsstat(1M)
fstab: file-system-table ..... fstab(4)
stat, lstat, fstat: get file status ..... stat(2)
statfs, fstatfs: get filesystem information ..... statfs(2)
identifier fstyp: determine filesystem ..... fstyp(1M)
fseek, rewind, ftell: reposition a file pointer in a stream ..... fseek(3S)
communication package stdipc: ftok: standard interprocess ..... stdipc(3C)
specified length truncate, ftruncate: truncate a file to a ..... truncate(2SEQ)
ftw: walk a file tree ..... ftw(3C)
function and complementary error function ..... erf(3M)
gamma: log gamma function ..... gamma(3M)
hypot: Euclidean distance function ..... hypot(3M)
entries of a common object file function ..... ldlread(3X)
matherr: error-handling function ..... matherr(3M)
prof: profile within a function ..... prof(5)
erf, erfc: error function and complementary error function ..... erf(3M)
j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
crypt: password and file encryption functions ..... crypt(3X)
logarithm, power, square root functions ..... exp(3M)
ceiling, remainder, absolute value functions ..... floor(3M)
or perform other defect management functions ..... format(1M)
sinh, cosh, tanh: hyperbolic functions ..... sinh(3M)
sysi86: machine specific functions ..... sysi86(2)
keys to internal menu system functions ..... syskeymap(4)
acos, atan, atan2: trigonometric functions ..... trig(3M)
math: math functions and constants ..... math(5)
intro: introduction to functions and libraries ..... intro(3)
300, 300s: handle special functions of DASI 300 and 300s terminals ..... 300(1)
hp: handle special functions of Hewlett-Packard terminals ..... hp(1)
450: handle special functions of the DASI 450 terminal ..... 450(1)
file or file structure fuser: identify processes using a ..... fuser(1M)
specified controlling terminal fvhangup: virtually hangup the ..... fvhangup(2SEQ)
fread, fwrite: binary input/output ..... fread(3S)
accounting records fwtmp, wtmpfix: manipulate connect ..... fwtmp(1M)
gamma: log gamma function ..... gamma(3M)
gamma: log gamma function ..... gamma(3M)

ecvt, fcvt, gcvt: convert floating-point number to string .....ecvt(3C)
command gencc: create a front-end to the cc.....gencc(1M)
command forms astgen: program for generating/modifying ASSIST menus or.....astgen(1)
fsirand: install random inode generation numbers.....fsirand(1M)
rand, srand: simple random-number generator .....rand(3C)
streamtty: Generic STREAMS tty driver interface.....streamtty(7)
tty: controlling terminal (gentty) interface .....tty(7)
getgeombyname: get disk geometry by name .....getgeombyname(3SEQ)
character or word from a stream getc, getchar, fgetc, getw: get.....getc(3S)
or word from a stream getc, getchar, fgetc, getw: get character .....getc(3S)
working directory getcwd: get pathname of current.....getcwd(3C)
put in a file system independent/ getdents: read directory entries and.....getdents(2)
descriptor table index getdtablemax: get max in-use.....getdtablemax(2SEQ)
size getdtablesize: get descriptor table.....getdtablesize(2SEQ)
getuid, geteuid, getgid, getegid: get real user, effective user, real/.....getuid(2)
name getenv: return value for environment.....getenv(3C)
user, effective user, real/ getuid, geteuid, getgid, getegid: get real.....getuid(2)
number getfrm: returns the current frame.....getfrm(1F)
geometry by name getgeombyname: get disk/.....getgeombyname(3SEQ)
getuid, geteuid, getgid, getegid: get real user, effective user,/.....getuid(2)
setgrent, endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, ...getgrent(3C)
endgrent, fgetgrent: get/ getgrent, getgrgid, setgrent,.....getgrent(3C)
fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent, endgrent, ...getgrent(3C)
IDs getgroups: get supplementary group.....getgroups(2)
currently marked menu items getitems: returns a list of the.....getitems(1F)
getlogin: get login name .....getlogin(3C)
file system/ addmntent, endmntent, getmntent, hasmntopt, setmntent: get....getmntent(3SEQ)
stream getmsg: get next message off a .....getmsg(2)
argument vector getopt: get option letter from.....getopt(3C)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
options getopts, getoptcvt: parse command.....getopts(1)
getpagesize: get system page size.....getpagesize(2SEQ)
getpass: read a password.....getpass(3C)
process group, and parent/ getpid, getpgrp, getppid: get process, ...getpid(2)
process, process group, and parent/ getpid, getpgrp, getppid: get.....getpid(2)
and parent process/ getpid, getpgrp, getppid: get process, process group, ...getpid(2)
utilization information get_process_stats: get resource....get_process_stats(2SEQ)
getpseudotty: get pseudo terminals.....getpseudotty(3SEQ)
getpw: get name from IUD .....getpw(3C)
setpwent, endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam,.....getpwent(3C)
fgetpwent: get/ getpwent, getpwuid, getpwnam, setpwent, endpwent,.....getpwent(3C)
endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam, setpwent,.....getpwent(3C)
stream gets, fgets: get a string from a.....gets(3S)
readfile, longline: reads file and gets longest line.....readfile(1F)
scsiinfo structure getscsiinfo, getscsimatch: get.....getscsiinfo(3SEQ)
getscsiinfo, getscsimatch: get scsiinfo structure.....getscsiinfo(3SEQ)
endspent, fgetspent, lckpwdf,/ getspent, getspnam, setspent,.....getspent(3X)
fgetspent, lckpwdf,/ getspent, getspnam, setspent, endspent,.....getspent(3X)
speed and terminal settings used by getty.....gettydefs(4)
speed, and line discipline getty: set terminal type, modes,.....getty(1M)
ct: spawn getty to a remote terminal.....ct(1C)

settings used by getty gettydefs: speed and terminal .....gettydefs(4)
get real user, effective user, real/ getuid, geteuid, getgid, getegid:.....getuid(2)
microsecond clock getusclk, usclk_init: get/init.....getusclk(3SEQ)
pututline, setutent, endutent,/ getut: getut: getutent, getutid, getutline,.....getut(3C)
pututline, setutent,/ getut: getutent, getutid, getutline,.....getut(3C)
getut: getutent, getutid, getutline, pututline, setutent,/.....getut(3C)
getut: getutent, getutid, getutline, pututline, setutent, endutent,/.....getut(3C)
/setutent, endutent, utmpname, getutpid: access utmp file entry.....getut(3C)
getc, getchar, fgetc, getw: get character or word from a stream .....getc(3S)
zdinfo structure getzdinfo, getzdinfobyname: get.....getzdinfo(3SEQ)
getzdinfo, getzdinfobyname: get zdinfo structure .....getzdinfo(3SEQ)
m_next: increment global counter .....m_next(3PPS)
task: global security table .....task(4)
operating system terms and symbols glossary: definitions of common.....glossary(1)
mktime, tzset:/ ctime, localtime, gmtime, asctime, cftime, ascftime,.....ctime(3C)
setjmp, longjmp: non-local goto.....setjmp(3C)
format of graphical files gps: graphical primitive string,.....gps(4)
primitive string, format of graphical files.....gps(4)
of graphical files gps: graphical primitive string, format.....gps(4)
plot: graphics interface .....plot(4)
plot: graphics interface subroutines.....plot(3X)
greek: select terminal filter .....greek(1)
grep: search a file for a pattern .....grep(1)
chown, chgrp: change owner or group.....chown(1)
newgrp: log in to a new group.....newgrp(1M)
run command in its own process group.....setpgrp(1)
initgroups: initialize group access list .....initgroups(3SEQ)
setgroups: set group access list .....setgroups(2SEQ)
/get real user, effective user, real group, and effective group IDs.....getuid(2)
/getppid: get process, process group, and parent process IDs .....getpid(2)
group: group file.....group(4)
pwck, grpck: password/ group file checkers.....pwck(1M)
setgrent, endgrent, fgetgrent: get group file entry .....getgrent(3C)
group: group file.....group(4)
setpgrp: set process group ID .....setpgrp(2)
create session and set process group ID .....setsid(2)
tcgetpgrp: get foreground process group ID.....tcgetpgrp(2)
tcsetpgrp: set foreground process group ID .....tcsetpgrp(2)
copy file and set mode, user, and group IDs.....cpset(1)
getgroups: get supplementary group IDs.....getgroups(2)
user, real group, and effective group IDs.....getuid(2)
setuid, setgid: set user and group IDs.....setuid(2)
id: print user and group IDs and names.....id(1M)
groups: show group memberships .....groups(1)
chown: change owner and group of a file.....chown(2)
send a signal to a process or a group of processes.....kill(2)
maintain, update, and regenerate groups of programs .....make(1)
groups: show group memberships .....groups(1)
pwck, grpck: password/group file checkers.....pwck(1M)
ssignal, gsignal: software signals.....ssignal(3C)
stty, gtty: set and get terminal state (defunct).....stty(2)
terminal fvhangup: virtually hangup the specified controlling.....fvhangup(2SEQ)

nohup: run a command immune to hangups and quits.....nohup(1)
hsearch, hcreate, hdestroy: manage hash search tables.....hsearch(3C)
spell, hashmake, spellin, hashcheck: find spelling errors .....spell(1)
crypt, setkey, encrypt: generate hashing encryption .....crypt(3C)
spelling errors spell, hashmake, spellin, hashcheck: find.....spell(1)
addmntent, endmntent, getmntent, hasmntopt, setmntent: get file/.....getmntent(3SEQ)
search tables hsearch, hcreate, hdestroy: manage hash .....hsearch(3C)
hsearch, hcreate, hdestroy: manage hash search tables .....hsearch(3C)
scnhdr: section header for a common object file.....scnhdr(4)
filehdr: file header for common object files.....filehdr(4)
limits: file header for implementation-specific constants.....limits(4)
unistd: file header for symbolic constants.....unistd(4)
ldfhread: read the file header of a common object file.....ldfhread(3X)
ldohseek: seek to the optional file header of a common object file .....ldohseek(3X)
/read an indexednamed section header of a common object file.....ldshread(3X)
ldahread: read the archive header of a member of an archive file.....ldahread(3X)
helplog: monitor help actions.....helplog(4)
help: operating system Help Facility.....help(1)
helpadm: change the Help Facility database .....helpadm(1M)
help: operating system Help Facility.....help(1)
database helpadm: change the Help Facility .....helpadm(1M)
helplog: monitor help actions.....helplog(4)
file archiver hpio: Hewlett-Packard 2645A terminal tape .....hpio(1)
hp: handle special functions of Hewlett-Packard terminals .....hp(1)
cronlog: cron history file.....cronlog(4)
ott: files that hold object architecture information.....ott(4)
uptime: show how long system has been up.....uptime(1)
Hewlett-Packard terminals hp: handle special functions of.....hp(1)
tape file archiver hpio: Hewlett-Packard 2645A terminal .....hpio(1)
hash search tables hsearch, hcreate, hdestroy: manage.....hsearch(3C)
sinh, cosh, tanh: hyperbolic functions .....sinh(3M)
hypot: Euclidean distance function .....hypot(3M)
u3b5, vax: get processor/ machid: i386, ns32000, pdp11, u3b, u3b2,.....machid(1)
consistency check icheck: filesystem storage.....icheck(1M)
ident: identify files.....ident(1)
m_get_myid: return process identification.....m_get_myid(3PPS)
issue: issue identification file.....issue(4)
fstyp: determine filesystem identifier.....fstyp(1M)
shmget: get shared memory segment identifier.....shmget(2)
file and set mode, user, and group IDs.....cpset(1)
getgroups: get supplementary group IDs.....getgroups(2)
process group, and parent process IDs.....getpid(2)
real group, and effective group IDs.....getuid(2)
setuid, setgid: set user and group IDs.....setuid(2)
id: print user and group IDs and names .....id(1M)
/fpsetmask, fpgetsticky, fpsetsticky: IEEE floating-point environment/.....fpgetround(3C)
core: format of core image file.....core(4)
scr_dump: format of curses screen image file .....scr_dump(4)
crash: examine system images.....crash(1M)
limits: file header for implementation-specific constants .....limits(4)
pwfix: remove incorrectly formatted password entries.....pwfix(1M)
m_next: increment global counter.....m_next(3PPS)

dump, dumpdates: incremental dump format.....dump(4)
dump: incremental filesystem dump.....dump(1M)
restore: incremental filesystem restore.....restore(1M)
get maximum in-use descriptor table index.....getdtablemax(2SEQ)
common/ ldtbindex: compute the index of a symbol table entry of a.....ldtbindex(3X)
common object/ ldtbread: read an indexed symbol table entry of a.....ldtbread(3X)
common/ldshread, ldnshread: read an indexednamed section header of a.....ldshread(3X)
ldsseek, ldnsseek: seek to an indexednamed section of a common/.....ldsseek(3X)
receipt of an orderly release indication.....t_rcvrel(3N)
receive a unit data error indication.....t_rcvuderr(3N)
specific alarms and/or the working indicator.....indicator(1F)
specific alarms and/or the working/ indicator: display application.....indicator(1F)
calculate number of blocks and indirect blocks.....nblocks(3SEQ)
syscall: indirect system call.....syscall(2)
terminfo descriptions infocmp: compare or print out.....infocmp(1M)
getusclk, usclk_init: get/ init microsecond clock.....getusclk(3SEQ)
inittab: script for the init process.....inittab(4)
initialization init, telinit: process control.....init(1M)
files directory init.d: system level transition.....init.d(4)
list initgroups: initialize group access.....initgroups(3SEQ)
init, telinit: process control initialization.....init(1M)
reinit: run an initialization file.....reinit(1F)
bcheckrc: system initialization procedures.....bcheckrc(1M)
tput: initialize a terminal or query terminfo database.....tput(1)
s_init_barrier, s_wait_barrier: initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
initgroups: initialize group access list.....initgroups(3SEQ)
/s_lock, s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
setup: initialize system for first user.....setup(1)
inittab: script for the init process.....inittab(4)
clri: clear inode.....clri(1M)
fs, inode: format of file system volume.....fs(4)
fsirand: install random inode generation numbers.....fsirand(1M)
number of free disk blocks and inodes.....df(1M)
fscanf, sscanf: convert formatted input.....scanf(3S)
ungetc: push character back into input stream.....ungetc(3S)
fread, fwrite: binary input/output.....fread(3S)
poll: STREAMS input/output multiplexing.....poll(2)
stdio: standard buffered input/output package.....stdio(3S)
uustat: uucp status inquiry and job control.....uustat(1C)
install: install commands.....install(1M)
install: install commands.....install(1M)
fsirand: install random inode generation numbers.....fsirand(1M)
on the BOM bomverify: perform an installation consistency check based.....bomverify(1M)
on a disk installboot: installs a boot program.....installboot(1M)
product: installed products file.....product(4)
installboot: installs a boot program on a disk.....installboot(1M)
with information from/ pwconv: installs and updates /etc/shadow.....pwconv(1M)
dis386: disassemble Series 386 instructions.....dis386(3SEQ)
mailx: interactive message processing system.....mailx(1)
filesystem consistency check and interactive repair.....fsck(1M)
console: console interface.....console(7)
lp: Systech parallel line printer interface.....lp(7)

mtio: UNIX magnetic tape interface ..... mtio(7)
plot: graphics interface ..... plot(4)
Generic STREAMS tty driver interface ..... streamtty(7)
swap: swap administrative interface ..... swap(1M)
termio: general terminal interface ..... termio(7)
tg: SCSI 1/2-inch magnetic tape interface ..... tg(7)
tty: controlling terminal (gentty) interface ..... tty(7)
wp: SSM parallel line printer interface ..... wp(7)
Xylogics 472 MULITBUS magnetic tape interface ..... xt(7)
timod: Transport Interface cooperating STREAMS module ..... timod(7)
STREAMS module tirdwr: Transport Interface read/write interface ..... tirdwr(7)
Transport Interface read/write interface STREAMS module ..... tirdwr(7)
plot: graphics interface subroutines ..... plot(3X)
sysadm: menu interface to do system administration ..... sysadm(1)
and event tracing log: interface to STREAMS error logging ..... log(7)
errorlog: internal error log file ..... errorlog(4)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
asa: interpret ASA carriage control characters ..... asa(1)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
csh: a shell (command interpreter) with C-like syntax ..... csh(1)
pipe: create an interprocess channel ..... pipe(2)
ipcs: report inter-process communication facilities status ..... ipcs(1)
stdipc: ftok: standard interprocess communication package ..... stdipc(3C)
tfipc: interprocess communication reporter ..... tfipc(1M)
ncheck: generate pathnames from i-numbers ..... ncheck(1M)
streamio: STREAMS ioctl commands ..... streamio(7)
ioctl: control device ..... ioctl(2)
abort: generate an IOT fault ..... abort(3C)
semaphore set, or shared memory ID ipcrm: remove a message queue, ... ipcrm(1)
communication facilities status ipcs: report inter-process ..... ipcs(1)
/isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct,/ ..... ctype(3C)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl,/ ..... ctype(3C)
/iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper, toascci,/ ..... ctype(3C)
ttyname, isatty: find name of a terminal ..... ttyname(3C)
/isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph,/ ..... ctype(3C)
isalpha, isalnum, isspace, iscntrl,/ isdigit, isxdigit, islower, isupper, ... ctype(3C)
/isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper,/ ..... ctype(3C)
file manager, updates text files in ISIS file format ..... isismgr(1M)
files in ISIS file format isismgr: ISIS file manager, updates text ..... isismgr(1M)
text files in ISIS file format isismgr: ISIS file manager, updates ..... isismgr(1M)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace,/ ..... ctype(3C)
floating-point NaN (Not-A-Number) isnan: isnan, isnanf: test for ..... isnan(3C)
isnan: isnand, isnanf: test for floating-point NaN/ ..... isnan(3C)
(Not-A-Number) isnan: isnand, isnanf: test for floating-point NaN ..... isnan(3C)
/isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower,/ ..... ctype(3C)
/isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii,/ ..... ctype(3C)
/islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint,/ ..... ctype(3C)
issue identification file ..... issue(4)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace,/ ..... ctype(3C)
isalnum, isspace, iscntrl,/ isdigit, isxdigit, islower, isupper, isalpha, ... ctype(3C)
getpw: get name from IUD ..... getpw(3C)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)

bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
uustat: uucp status inquiry and job control ..... uustat(1C)
proto: prototype job file for at ..... proto(4)
join: relational database operator ..... join(1)
/erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48:/ ..... drand48(3C)
sigsetjmp, siglongjmp: non-local jumps ..... sigsetjmp(3C)
dumpconf: dump kernel auto-configuration device table ..... dumpconf(1M)
makekey: generate encryption key ..... makekey(1)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
apropos: locate commands by keyword lookup ..... apropos(1)
system command locate: use keywords to identify an operating ..... locate(1)
killall: kill all active processes ..... killall(1M)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
a group of processes kill: send a signal to a process or ..... kill(2)
kill: terminate a process ..... kill(1)
killall: kill all active processes ..... killall(1M)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
mem, kmem: core memory ..... mem(7)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
ksh, rksh: KornShell, a standard/restricted command and /..... ksh(1)
standard/restricted command and/ ksh, rksh: KornShell, a ..... ksh(1)
integers and long integers l3tol, ltol3: convert between 3-byte ..... l3tol(3C)
and base-64 ASCII string a64l, l64a: convert between long integer ..... a64l(3C)
filesystems labelit: provide labels for ..... labelit(1M)
labelit: provide labels for filesystems ..... labelit(1M)
awk: pattern scanning and processing language ..... awk(1)
bc: arbitrary-precision arithmetic language ..... bc(1)
efl: Extended Fortran Language ..... efl(1)
command and programming language ..... ksh(1)
pattern scanning and processing language ..... nawk(1)
command programming language ..... sh(1)
cftime: language specific strings ..... cftime(4)
chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp,/ ..... acctsh(1M)
/setspent, endspent, fgetspent, lckpwdf, ulckpwdf: get shadow/ ..... getspent(3X)
/mrand48, jrand48, srand48, seed48, lcong48: generate uniformly/ ..... drand48(3C)
files ld: link editor for object ..... see *C and Lang Tools Binder*
ld0: STREAMS tty line discipline ..... ld0(7)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
a member of an archive file ldahread: read the archive header of ..... ldahread(3X)
for reading ldopen, ldaopen: open a common object file ..... ldopen(3X)
object file ldclose, ldaclose: close a common ..... ldclose(3X)
floating-point numbers frexp, ldexp, modf: manipulate parts of ..... frexp(3C)
routines ldfcn: common object file access ..... ldfcn(4)
common object file ldfhread: read the file header of a ..... ldfhread(3X)
common object file symbol table/ ldgetname: retrieve symbol name for ..... ldgetname(3X)
number entries of a common/ldlread, ldlinit, ldlitem: manipulate line ..... ldlread(3X)
entries of a/ ldlread, ldlinit, ldlitem: manipulate line number ..... ldlread(3X)
manipulate line number entries of a/ ldlread, ldlinit, ldlitem: ..... ldlread(3X)
number entries of a section of a/ ldlseek, ldnlseek: seek to line ..... ldlseek(3X)
entries of a section of a/ ldlseek, ldnlseek: seek to line number ..... ldlseek(3X)
of a section of a common/ ldrseek, ldnrseek: seek to relocation entries ..... ldrseek(3X)

section header of a/ ldshread, ldnshread: read an indexednamed.....ldshread(3X)
section of a common object/ ldsseek, ldnsseek: seek to an indexednamed.....ldsseek(3X)
header of a common object file ldohseek: seek to the optional file.....ldohseek(3X)
object file for reading ldopen, ldaopen: open a common.....ldopen(3X)
relocation entries of a section of/ ldrseek, ldnrseek: seek to.....ldrseek(3X)
indexednamed section header of a/ ldshread, ldnshread: read an.....ldshread(3X)
indexednamed section of a common/ ldsseek, ldnsseek: seek to an.....ldsseek(3X)
symbol table entry of a common/ ldtbindex: compute the index of a.....ldtbindex(3X)
table entry of a common object file ldtbread: read an indexed symbol.....ldtbread(3X)
of a common object file ldtbseek: seek to the symbol table.....ldtbseek(3X)
truncate a file to a specified length.....truncate(2SEQ)
init.d: system level transition files directory.....init.d(4)
lexical tasks lex: generate programs for simple.....lex(1)
lex: generate programs for simple lexical tasks.....lex(1)
lsearch, lfind: linear search and update.....lsearch(3C)
intro: introduction to functions and libraries.....intro(3)
find ordering relation for an object library.....see *C and Lang Tools Binder*
introduction to Parallel Programming Library.....intro(3PPS)
t_sync: synchronize transport library.....t_sync(3N)
t_alloc: allocate a library structure.....t_alloc(3N)
t_free: free a library structure.....t_free(3N)
shell (command interpreter) with C- like syntax.....csh(1)
quota: display disk usage and limits.....quota(1)
ulimit: get and set user limits.....ulimit(2)
implementation-specific constants limits: file header for.....limits(4)
returns the number of CPUs on- line.....cpus_online(3PPS)
line: read one line.....line(1)
puts its arguments on message line.....message(1F)
all but one active processor off line.....offline_all(3SEQ)
reads file and gets longest line.....readfile(1F)
establish an out-going terminal line connection.....dial(3C)
set terminal type, modes, speed, and line discipline.....getty(1M)
ld0: STREAMS tty line discipline.....ld0(7)
set terminal type, modes, speed, and line discipline.....uugetty(1M)
man: display on- line man pages.....man(1)
linenum: line number entries in a common object file.....linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common object/.....ldlread(3X)
a common/ ldlseek, ldnlseek: seek to line number entries of a section of.....ldlseek(3X)
nl: line numbering filter.....nl(1)
cut: cut out selected fields of each line of a file.....cut(1)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
line: read one line.....line(1)
lsearch, lfind: linear search and update.....lsearch(3C)
col: filter reverse line-feeds.....col(1)
common object file linenum: line number entries in a.....linenum(4)
readlink: read value of a symbolic link.....readlink(2SEQ)
link, unlink: link and unlink files and directories.....link(1M)
ld: link editor for object files.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
link: link to a file.....link(2)
cp, ln, mv: copy, link or move files.....cp(1)

link: link to a file ..... link(2)
symlink: make symbolic link to a file ..... symlink(2SEQ)
and directories link, unlink: link and unlink files ..... link(1M)
lint: a C program checker ..... see *C and Lang Tools Binder*
initgroups: initialize group access list ..... initgroups(3SEQ)
nlist: get entries from name list ..... nlist(3C)
setgroups: set group access list ..... setgroups(2SEQ)
varargs: handle variable argument list ..... varargs(5)
output of a varargs argument list ..... vprintf(3S)
ls: list contents of directory ..... ls(1)
ff: list filenames and statistics for a filesystem ..... ff(1M)
fsck and ncheck checklist: list of filesystems processed by ..... checklist(4)
getitems: returns a list of the currently marked menu items ..... getitems(1F)
t_listen: listen for a connect request ..... t_listen(3N)
listing list: produce C source ..... see *C and Lang Tools Binder*
xargs: construct argument list(s) and execute command ..... xargs(1)
volcopy: make literal copy of filesystem ..... volcopy(1M)
cp, ln, mv: copy, link or move files ..... cp(1)
ascftime, mktime, tzset:/ ctime, localtime, gmtime, asctime, cftime, ... ctime(3C)
apropos: locate commands by keyword lookup ..... apropos(1)
operating system command locate: use keywords to identify an ..... locate(1)
setlocale: set programs location ..... setlocale(3C)
end, etext, edata: last locations in program ..... end(3C)
plock: lock process, text, or data in memory ..... plock(2)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
s_clock, s_unlock: initialize, lock, unlock locks ..... s_lock(3PPS)
lockf: record locking on files ..... lockf(3C)
lockf: record locking on files ..... lockf(3C)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
s_unlock: initialize, lock, unlock locks ..... s_lock(3PPS)
diagnostic messages to form error log ..... dmesg(1M)
errorlog: internal error log file ..... errorlog(4)
gamma: log gamma function ..... gamma(3M)
newgrp: log in to a new group ..... newgrp(1M)
logging and event tracing log: interface to STREAMS error ..... log(7)
logarithm, power, square root/ exp, log, log10, pow, sqrt: exponential, ... exp(3M)
about RCS files rlog: print log messages and other information ..... rlog(1)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
sulog: log of substitute user events ..... sulog(4)
logarithm, power, square/ exp, log, log10, pow, sqrt: exponential, ... exp(3M)
log, log10, pow, sqrt: exponential, logarithm, power, square root/ ..... exp(3M)
strclean: STREAMS error logger cleanup program ..... strclean(1M)
strerr: STREAMS error logger daemon ..... strerr(1M)
log: interface to STREAMS error logging and event tracing ..... log(7)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
getlogin: get login name ..... getlogin(3C)
logname: get login name ..... logname(1)
cuserid: get character login name of the user ..... cuserid(3S)
logname: return login name of user ..... logname(3X)
passwd: change login password and password attributes ..... passwd(1)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
login: sign on ..... login(1)

setting up an environment at login time.....profile(4)
/usr/adm/ loginlog: log of failed login attempts.....loginlog(4)
logname: get login name.....logname(1)
logname: return login name of user.....logname(3X)
setjmp, longjmp: non-local goto.....setjmp(3C)
readfile, longline: reads file and gets longest line.....readfile(1F)
apropos: locate commands by keyword lookup.....apropos(1)
make a database for fast password lookup.....mkpwdbm(1M)
relation for an object library lorder: find ordering.....see *C and Lang Tools Binder*
the LP print service lp, cancel: send/cancel requests to.....lp(1)
cancel: send/cancel requests to the LP print service.....lp(1)
lpadmin: configure the LP print service.....lpadmin(1M)
administer filters used with the LP print service.....lpfilter(1M)
administer forms used with the LP print service.....lpforms(1M)
information about the status of the LP print service.....lpstat(1)
/lpshut, lpmove: start/stop the LP print service and move requests.....lpsched(1M)
enable, disable: enable/disable LP printers.....enable(1)
accept, reject: allow or prevent LP requests.....accept(1M)
interface lp: Systech parallel line printer.....lp(7)
service lpadmin: configure the LP print.....lpadmin(1M)
with the LP print service lpfilter: administer filters used.....lpfilter(1M)
the LP print service lpforms: administer forms used with.....lpforms(1M)
processing module lpmod: parallel printer canonical.....lpmod(7)
service and move/ lpsched, lpshut, lpmove: start/stop the LP print.....lpsched(1M)
the LP print service and move/ lpsched, lpshut, lpmove: start/stop.....lpsched(1M)
print service and move/ lpsched, lpshut, lpmove: start/stop the LP.....lpsched(1M)
status of the LP print service lpstat: print information about the.....lpstat(1)
supported by the menu system lptypes: table of printer types.....lptypes(4)
priorities lpusers: set printing queue.....lpusers(1M)
srand48, seed48,/ drand48, erand48, lrand48, nrand48, mrand48, jrand48,.....drand48(3C)
ls: list contents of directory.....ls(1)
update lsearch, lfind: linear search and.....lsearch(3C)
lseek: move read/write file pointer.....lseek(2)
stat, lstat, fstat: get file status.....stat(2)
integers and long integers l3tol, ltol3: convert between 3-byte.....l3tol(3C)
m4: macro processor.....m4(1)
u3b2, u3b5, vax: get processor type/ machid: i386, ns32000, pdp11, u3b,.....machid(1)
show the configuration of the machine.....showcfg(1M)
sysi86: machine specific functions.....sysi86(2)
values: machine-dependent values.....values(5)
sgetl: access long integer data in a machine-independent fashion.....sputl(3X)
m4: macro processor.....m4(1)
mtio: UNIX magnetic tape interface.....mtio(7)
tg: SCSI 1/2-inch magnetic tape interface.....tg(7)
xt: Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
mt: magnetic tape manipulating program.....mt(1)
rmail: send mail to users or read mail.....mail(1)
read mail mail, rmail: send mail to users or.....mail(1)
mail, rmail: send mail to users or read mail.....mail(1)
enhancement request mailbug: submit a bug report or.....mailbug(1)
processing system mailx: interactive message.....mailx(1)
groups of programs make: maintain, update, and regenerate.....make(1)

programs intro: introduction to ..... maintenance commands and application ..... intro(1)
/ttyname_to_dev: convert between ..... major/minor numbers/tty names ..... dev_to_ttyname(3C)
regenerate groups of programs ..... make: maintain, update, and ..... make(1)
man page databases ..... makealiases, makewhatis: generate ..... makealiases(1M)
makedev: make system special files ..... makedev(1M)
makekey: generate encryption key ..... makekey(1)
databases makealiases, ... makewhatis: generate man page ..... makealiases(1M)
/free, realloc, calloc, mallopt, ... mallinfo: fast main memory allocator ..... malloc(3X)
memory allocator ..... malloc, free, realloc, calloc: main ..... malloc(3C)
mallopt, mallinfo: fast main memory/ ..... malloc, free, realloc, calloc, ... malloc(3X)
malloc, free, realloc, calloc, ... mallopt, mallinfo: fast main memory/ ..... malloc(3X)
man: display on-line man pages ..... man(1)
makealiases, makewhatis: generate ..... man page databases ..... makealiases(1M)
man: display on-line ..... man pages ..... man(1)
tsearch, tfind, tdelete, twalk: ..... manage binary search trees ..... tsearch(3C)
hsearch, hcreate, hdestroy: ..... manage hash search tables ..... hsearch(3C)
endpoint t_optmgmt: ..... manage options for a transport ..... t_optmgmt(3N)
passmgmt: password files ..... management ..... passmgmt(1M)
sigignore, sigpause: signal ..... management ..... sigset(2)
format disks or perform other defect ..... management functions ..... format(1M)
file format isismgr: ISIS file ..... manager, updates text files in ISIS ..... isismgr(1M)
mt: magnetic tape ..... manipulating program ..... mt(1)
parameters of virtual-to-physical ..... mapping devices ..... pmap(1M)
m_single, m_multi: ..... mark single-process code section ..... m_single(3PPS)
umask: set file-creation mode ..... mask ..... umask(1)
umask: set and get file creation ..... mask ..... umask(2)
regex: ..... match patterns against a string ..... regex(1F)
regular expression compile and ..... match routines ..... regexp(5)
math: ..... math functions and constants ..... math(5)
math: math functions and constants ..... math(5)
matherr: error-handling function ..... matherr(3M)
mbad: MULITBUS adapter ..... mbad(7)
comment section ..... mcs: manipulate the object file ..... mcs(1)
mem, kmem: core memory ..... mem(7)
read the archive header of a ..... member of an archive file ..... ldahread(3X)
groups: show group ..... memberships ..... groups(1)
memset: memory operations ..... memccpy, memchr, memcmp, memcpy, ... memory(3C)
memory operations ..... memchr, memcmp, memcpy, memset: ..... memory(3C)
memory: memccpy, memchr, memcmp, ... memcmp, memcpy, memset: memory ..... memory(3C)
kMBmem, kMWmem: MULITBUS ..... memcpy, memset: memory operations ..... memory(3C)
mem, kmem: core ..... memory ..... kmmem(7)
lock process, text, or data in ..... memory ..... mem(7)
create security table in shared ..... memory ..... plock(2)
malloc, free, realloc, calloc: main ..... memory ..... tasktbl(1M)
calloc, mallopt, mallinfo: fast main ..... memory allocator ..... malloc(3C)
shmalloc, shrealloc, shfree: shared ..... memory allocator ..... malloc(3X)
valloc: aligned ..... memory allocator ..... shmalloc(3PPS)
shmctl: shared ..... memory allocator ..... valloc(3SEQ)
consmem, sm, ss: console ..... memory control operations ..... shmctl(2)
dump: standalone ..... memory drivers ..... consmem(7)
spawn new process in a virtual ..... memory dump ..... dump(8)
memory efficient way ..... vfork(2)

queue, semaphore set, or shared memory ID ..... ipcrm(1)
memcpy, memset: memory operations memory: memccpy, memchr, memcmp, ... memory(3C)
memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
vmtune: tune system-wide virtual- memory parameters ..... vmtune(1M)
shmget: get shared memory segment identifier ..... shmget(2)
vm_ctl: examine and change virtual memory tuning parameters ..... vm_ctl(2SEQ)
memccpy, memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
addcmdopt: add a command option to a menu ..... addcmdopt(1M)
add a form/SELECT window option to a menu ..... addformopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
sysadm: menu interface to do system administration ..... sysadm(1)
a list of the currently marked menu items ..... getitems(1F)
menu: menu processor ..... menu(1M)
menumsg: route user message to menu message queue ..... menumsg(1M)
rmvopt: remove a menu option ..... rmvopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
menu: menu processor ..... menu(1M)
of printer types supported by the menu system ..... lptypes(4)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
message queue menumsg: route user message to menu ..... menumsg(1M)
for generating/modifying ASSIST menus or command forms ..... astgen(1)
merge: three-way file merge ..... merge(1)
sort: sort and/or merge files ..... sort(1)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
subsequent lines of one file paste: merge same lines of several files or ..... paste(1)
merge: three-way file merge ..... merge(1)
mesg: permit or deny messages ..... mesg(1)
t_error: produce error message ..... t_error(3N)
tfnotify: send completion message ..... tfnotify(1M)
msgctl: message control operations ..... msgctl(2)
message: puts its arguments on message line ..... message(1F)
getmsg: get next message off a stream ..... getmsg(2)
putmsg: send a message on a stream ..... putmsg(2)
msgop: message operations ..... msgop(2)
mailx: interactive message processing system ..... mailx(1)
message line message: puts its arguments on ..... message(1F)
menumsg: route user message to menu message queue ..... menumsg(1M)
msgget: get message queue ..... msgget(2)
shared memory ID ipcrm: remove a message queue, semaphore set, or ..... ipcrm(1)
errormsg: error message text file ..... errormsg(4)
menumsg: route user message to menu message queue ..... menumsg(1M)
motd: message-of-the-day file ..... motd(4)
mesg: permit or deny messages ..... mesg(1)
sys_errlist, sys_nerr: system error messages ..... perror(3C)
strace: print STREAMS trace messages ..... strace(1M)
rlog: print log messages and other information about RCS files ..... rlog(1)
dmesg: collect system diagnostic messages to form error log ..... dmesg(1M)
parallel m_fork: execute a subprogram in ..... m_fork(3PPS)
identification m_get_myid: return process ..... m_get_myid(3PPS)

number of child processes — m_get_numprocs: return ..... m_get_numprocs(3PPS)
getusclk, usclk_init: get/init — microsecond clock ..... getusclk(3SEQ)
usclk: 32-bit — microsecond counter ..... usclk(7)
aging, PFF, swapping, or processor — migration disabled ..... noage(1M)
clone: open any — minor device on a STREAMS driver ..... clone(7)
/convert between major/ — minor numbers and tty names ..... dev_to_ttyname(3C)
/acctwtmp: overview of accounting and — miscellaneous accounting commands ..... acct(1M)
intro: introduction to — miscellany ..... intro(5)
mkdir: make a directory ..... mkdir(2)
mkdir: make directories ..... mkdir(1)
mkfifo: make a FIFO special file ..... mkfifo(3C)
mkfs: construct a filesystem ..... mkfs(1M)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
mknod: build special file ..... mknod(1M)
special or ordinary file — mknod: make a directory, or a ..... mknod(2)
password lookup — mkpwdbm: make a database for fast ..... mkpwdbm(1M)
mktemp: make a unique filename ..... mktemp(3C)
/gmtime, asctime, cftime, ascftime, — mktime, tzset: convert date and time/ ..... ctime(3C)
mkvtoc: populate VTOC on disks ..... mkvtoc(1M)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
processs address space — mmap: map an open file into the ..... mmap(2SEQ)
section m_single, — m_multi: mark single-process code ..... m_single(3PPS)
m_next: increment global counter ..... m_next(3PPS)
mnttab: mounted filesystem table ..... mnttab(4)
configuration file — mod_config: format of a module ..... mod_config(4)
chmod: change — mode ..... chmod(1)
umask: set file-creation — mode mask ..... umask(1)
chmod: change — mode of file ..... chmod(2)
cpset: copy file and set — mode, user, and group IDs ..... cpset(1)
getty: set terminal type, — modes, speed, and line discipline ..... getty(1M)
uugetty: set terminal type, — modes, speed, and line discipline ..... uugetty(1M)
frexp, ldexp, — modf: manipulate parts of floating-point/ ..... frexp(3C)
utime: set file access and — modification times ..... utime(2)
touch: update access and — modification times of a file ..... touch(1)
astgen: program for generating/ — modifying ASSIST menus or command/ ..... astgen(1)
printer canonical processing — module ..... lpmod(7)
SSM: System Services — Module ..... ssm(7)
Interface cooperating STREAMS — module ..... timod(7)
read/write interface STREAMS — module ..... tirdwr(7)
mod_config: format of a — module configuration file ..... mod_config(4)
/ckpacct, dodisk, lastlogin, — monacct, nulladm, prctmp, prdaily,/ ..... acctsh(1M)
helplog: — monitor help actions ..... helplog(4)
monitor: monitor system activity ..... monitor(1M)
monitor: prepare execution profile ..... monitor(3C)
monitor: — monitor system activity ..... monitor(1M)
a time — more: view a file one full screen at ..... more(1)
motd: message-of-the-day file ..... motd(4)
mount: — mount a filesystem ..... mount(2)
mount, umount: — mount and unmount filesystems ..... mount(1M)
mount: mount a filesystem ..... mount(2)
setmnt: establish — mount table ..... setmnt(1M)
filesystems — mount, umount: mount and unmount ..... mount(1M)

mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
multiple filesystems mountall, umountall: mount, unmount.....mountall(1M)
mnttab: mounted filesystem table.....mnttab(4)
mvdir: move a directory.....mvdir(1M)
cp, ln, mv: copy, link or move files.....cp(1)
lseek: move read/write file pointer.....lseek(2)
start/stop the LP print service and move requests.....lpsched(1M)
and resume child process execution m_park_procs, m_rele_procs: susp/....m_park_procs(3PPS)
ptrace, mptrace: process trace facility.....ptrace(2)
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48,/.....drand48(3C)
child process/ m_park_procs, m_rele_procs: suspend and resume....m_park_procs(3PPS)
processes m_set_procs: set number of child.....m_set_procs(3PPS)
msgctl: message control operations.....msgctl(2)
msgget: get message queue.....msgget(2)
msgop: message operations.....msgop(2)
single-process code section m_single, m_multi: mark.....m_single(3PPS)
m_sync: check in at barrier.....m_sync(3PPS)
program mt: magnetic tape manipulating.....mt(1)
st: Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
mtio: UNIX magnetic tape interface.....mtio(7)
mbad: MULITBUS adapter.....mbad(7)
xt: Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
kMBmem, kMWmem: MULITBUS memory.....kmmem(7)
mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
poll: STREAMS input/output multiplexing.....poll(2)
Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
tmp_ctl: multi-processor control.....tmp_ctl(2SEQ)
rc2: run commands performed for multi-user environment.....rc2(1M)
m_lock, m_unlock: lock, unlock locks.....m_lock(3PPS)
address space munmap: unmap part of the processs.....munmap(2SEQ)
cp, ln, mv: copy, link or move files.....cp(1)
mvdir: move a directory.....mvdir(1M)
devnm: device name.....devnm(1M)
getenv: return value for environment name.....getenv(3C)
getgeombyname: get disk geometry by name.....getgeombyname(3SEQ)
getlogin: get login name.....getlogin(3C)
logname: get login name.....logname(1)
pwd: working directory name.....pwd(1)
tmpnam, tempnam: create a name for a temporary file.....tmpnam(3S)
table/ ldgetname: retrieve symbol name for common object file symbol.....ldgetname(3X)
getpw: get name from IUD.....getpw(3C)
nlist: get entries from name list.....nlist(3C)
rename: change the name of a file.....rename(2)
ttyname, isatty: find name of a terminal.....ttyname(3C)
uname: get name of current operating system.....uname(2)
uname: print name of current system.....uname(1)
tty: get the name of the terminal.....tty(1)
cuserid: get character login name of the user.....cuserid(3S)
logname: return login name of user.....logname(3X)
/ldnshread: read an indexed named section header of a common object/.....ldshread(3X)
/ldnsseek: seek to an indexed named section of a common object file.....ldsseek(3X)

between major/minor numbers and tty names.....dev_to_ttyname(3C)
id: print user and group IDs and names.....id(1M)
term: conventional names for terminals.....term(5)
isnanf: test for floating-point NaN (Not-A-Number).....isnan(3C)
processing language nawk: pattern scanning and .....nawk(1)
and indirect blocks nblocks: calculate number of blocks.....nblocks(3SEQ)
of filesystems processed by fsck and ncheck.....checklist(4)
i-numbers ncheck: generate pathnames from .....ncheck(1M)
uucp: networking support files.....uucp(4)
file newform: change the format of a text .....newform(1)
newfs: construct a new file system .....newfs(1M)
newgrp: log in to a new group.....newgrp(1M)
news: print news items.....news(1)
news: print news items.....news(1)
getmsg: get next message off a stream.....getmsg(2)
nice: change priority of a process.....nice(2)
nice: run a command at low priority .....nice(1)
nl: line numbering filter.....nl(1)
nlist: get entries from name list.....nlist(3C)
object file nm: print name list of.....see *C and Lang Tools Binder*
command with priority aging, PFF,/ noage, nopff, noswap, onproc: run a.....noage(1M)
hangups and quits nohup: run a command immune to.....nohup(1)
setjmp, longjmp: non-local goto.....setjmp(3C)
sigsetjmp, siglongjmp: non-local jumps.....sigsetjmp(3C)
with priority aging, PFF,/ noage, nopff, noswap, onproc: run a command.....noage(1M)
priority aging, PFF,/ noage, nopff, noswap, onproc: run a command with .....noage(1M)
test for floating-point NaN (Not-A-Number).....isnan(3C)
seed48,/ drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, ...drand48(3C)
deroff: remove nroff, tbl, and eqn constructs.....deroff(1)
vax: get processor/ machid: i386, ns32000, pdp11, u3b, u3b2, u3b5,.....machid(1)
null: the null file.....null(7)
null: the null file .....null(7)
/ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct,/ .....acctsh(1M)
obtain the prime factors of a number.....factor(1)
getfrm: returns the current frame number.....getfrm(1F)
test for floating-point NaN (Not-A- Number) .....isnan(3C)
convert string to double-precision number.....strtod(3C)
linenum: line number entries in a common object file.....linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common object/.....ldlread(3X)
ldlseek, ldnlseek: seek to line number entries of a section of a/ .....ldlseek(3X)
rand, srand: simple random- number generator.....rand(3C)
nblocks: calculate number of blocks and indirect blocks.....nblocks(3SEQ)
m_get_numprocs: return number of child processes .....m_get_numprocs(3PPS)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
cpus_online: returns the number of CPUs on-line .....cpus_online(3PPS)
df: report number of free disk blocks and inodes.....df(1M)
fcvt, gcvt: convert floating-point number to string .....ecvt(3C)
nl: line numbering filter.....nl(1)
uniformly distributed pseudorandom numbers.....drand48(3C)
manipulate parts of floating-point numbers .....frexp(3C)
install random inode generation numbers .....fsirand(1M)
to system calls and error numbers .....intro(2)

ncheck: generate pathnames from i- numbers.....ncheck(1M)
/convert between major/minor numbers and tty names.....dev_to_ttyname(3C)
cprs: compress an object file.....see *C and Lang Tools Binder*
strstr: find first occurrence of character.....strstr(3C)
od: octal dump.....od(1)
od: octal dump.....od(1)
offline: bring processors online and offline.....online(1M)
online, offline: bring processors online and offline.....online(1M)
processor off line offline_all: take all but one active.....offline_all(3SEQ)
online, offline: bring processors online and offline.....online(1M)
online, offline: bring processors online, offline: bring processors.....online(1M)
aging, PFF/ noage, nopff, noswap, onproc: run a command with priority.....noage(1M)
ldopen, ldaopen: open a common object file for reading.....ldopen(3X)
fopen, freopen, fdopen: open a stream.....fopen(3S)
clone: open any minor device on a STREAMS driver.....clone(7)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
mmap: map an open file into the process address space.....mmap(2SEQ)
open: open for reading or writing.....open(2)
open: open for reading or writing.....open(2)
rewinddir, closedir/ directory: opendir, readdir, telldir, seekdir,.....directory(3X)
a standalone program or bring up the operating system.....boot(8)
run commands performed to stop the operating system.....rc0(1M)
savecore: save a core dump of the operating system.....savecore(1M)
uname: get name of current operating system.....uname(2)
locate: use keywords to identify an operating system command.....locate(1)
assist: assistance using operating system commands.....assist(1)
starter: information about the operating system for beginning users.....starter(1)
help: operating system Help Facility.....help(1)
glossary: definitions of common operating system terms and symbols.....glossary(1)
join: relational database operator.....join(1)
curses: terminal screen handling and optimization package.....curses(3X)
rmvopt: remove a menu option.....rmvopt(1M)
getopt: get option letter from argument vector.....getopt(3C)
addcmdopt: add a command option to a menu.....addcmdopt(1M)
addformopt: add a form/SELECT window option to a menu.....addformopt(1M)
addmenuopt: add a menu option to a menu.....addmenuopt(1M)
object file ldohseek: seek to the optional file header of a common.....ldohseek(3X)
fcntl: file control options.....fcntl(5)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
stty: set the options for a terminal.....stty(1)
t_optmgmt: manage options for a transport endpoint.....t_optmgmt(3N)
make a directory, or a special or ordinary file.....mknod(2)
architecture information ott: files that hold object.....ott(4)
common assembler and link editor output.....a.out(4)
echo: put string on virtual output.....echo(1F)
fread, fwrite: binary input/ output.....fread(3S)
fprintf, sprintf: print formatted output.....printf(3S)
poll: STREAMS input/ output multiplexing.....poll(2)
/vfprintf, vsprintf: print formatted output of a varargs argument list.....vprintf(3S)
stdio: standard buffered input/ output package.....stdio(3S)

setpgrp: run command in its own process group....setpgrp(1)
chown: change owner and group of a file....chown(2)
chown, chgrp: change owner or group....chown(1)
expand files pack, pcat, unpack: compress and....pack(1)
makewhatis: generate man page databases....makealiases(1M)
getpagesize: get system page size....getpagesize(2SEQ)
4014: paginator for the Tektronix 4014 terminal....4014(1)
m_fork: execute a subprogram in parallel....m_fork(3PPS)
lp: Systech parallel line printer interface....lp(7)
wp: SSM parallel line printer interface....wp(7)
lpmod: parallel printer canonical processing module....lpmod(7)
intro: introduction to Parallel Programming Library....intro(3PPS)
and change virtual memory tuning parameters....vm_ctl(2SEQ)
tune system-wide virtual-memory parameters....vmtune(1M)
pmap: display or alter parameters of virtual-to-physical mapping/....pmap(1M)
get process, process group, and parent process IDs....getpid(2)
getopt: parse command options....getopt(1)
getopts, getoptcvt: parse command options....getopts(1)
cmpt: compatibility partitioning driver....cmpt(7)
vtoc: VTOC partitioning driver....vtoc(7)
frexp, ldexp, modf: manipulate parts of floating-point numbers....frexp(3C)
passmgmt: password files management....passmgmt(1M)
with information from /etc/ passwd....pwconv(1M)
password attributes passwd: change login password and....passwd(1)
passwd: password file....passwd(4)
getpass: read a password....getpass(3C)
crypt: password and file encryption functions....crypt(3X)
passwd: change login password and password attributes....passwd(1)
passwd: change login password and password attributes....passwd(1)
pwdhd: daemon to oversee password database creation....pwdhd(1M)
pwfix: remove incorrectly formatted password entries....pwfix(1M)
passwd: password file....passwd(4)
shadow: restricted password file....shadow(4)
setpwent, endpwent, fgetpwent: get password file entry....getpwent(3C)
lckpwdf, ulckpwdf: get shadow password file entry....getspent(3X)
putpwent: write password file entry....putpwent(3C)
putspent: write shadow password file entry....putspent(3X)
converts from a two to a one password file scheme....pwunconv(1M)
passmgmt: password files management....passmgmt(1M)
mkpwdbm: make a database for fast password lookup....mkpwdbm(1M)
pwck, grpck: password/group file checkers....pwck(1M)
files or subsequent lines of one/ paste: merge same lines of several....paste(1)
bp: binary patcher....bp(1M)
pathalias: alias file for FACE....alias(4)
configurable pathname variables pathconf, fpathconf: get....pathconf(2)
criteria for filename pathconv: search Interpreter....pathconv(1F)
getcwd: get pathname of current working directory....getcwd(3C)
fpathconf: get configurable pathname variables....pathconf(2)
dirname: deliver portions of pathnames....basename(1)
ncheck: generate pathnames from i-numbers....ncheck(1M)
sweepon, sweepoff: enable/disable patrol seeks on DCC disks....sweepon(1M)
grep: search a file for a pattern....grep(1)

awk: pattern scanning and processing language ..... awk(1)
nawk: pattern scanning and processing language ..... nawk(1)
egrep: search a file for a pattern using full regular/..... egrep(1)
regex: match patterns against a string ..... regex(1F)
pause: suspend process until signal ..... pause(2)
pack, pcat, unpack: compress and expand files ..... pack(1)
popen, pclose: initiate pipe to/from a process ..... popen(3S)
pdbx, dbx: parallel/..... see *C and Lang Tools Binder*
machid: i386, ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor/..... machid(1)
sigpending: examine pending signals ..... sigpending(2)
check the uucp directories and permissions file..... uucheck(1M)
acct: per-process accounting file format ..... acct(4)
acctcms: command summary from per-process accounting records ..... acctcms(1M)
sys_nerr: system error messages perror, errno, sys_errlist, ... perror(3C)
run a command with priority aging, PFF, swapping, or processor/..... noage(1M)
pg: file perusal filter for CRTs ..... pg(1)
virtual address space pmap: map physical addresses into processs ..... pmap(7)
or alter parameters of virtual-to- physical mapping devices ..... pmap(1M)
pipe: create an interprocess channel ..... pipe(2)
sp: STREAMS pipe device ..... sp(7)
tee: pipe fitting ..... tee(1)
popen, pclose: initiate pipe to/from a process ..... popen(3S)
in memory plock: lock process, text, or data ..... plock(2)
plot: graphics interface ..... plot(4)
plot: graphics interface subroutines ..... plot(3X)
virtual-to-physical mapping devices pmap: display or alter parameters of..... pmap(1M)
processs virtual address space pmap: map physical addresses into ..... pmap(7)
/fpsetsticky: IEEE floating- point environment control ..... fpgetround(3C)
isnand, isnanf: test for floating- point NaN (Not-A-Number) ..... isnan(3C)
ecvt, fcvt, gcvt: convert floating- point number to string ..... ecvt(3C)
modf: manipulate parts of floating- point numbers ..... frexp(3C)
lseek: move read/write file pointer..... lseek(2)
rewind, ftell: reposition a file pointer in a stream..... fseek(3S)
multiplexing poll: STREAMS input/output ..... poll(2)
a process popen, pclose: initiate pipe to/from ..... popen(3S)
banner: make posters ..... banner(1)
power, square root/ exp, log, log10, pow, sqrt: exponential, logarithm, ... exp(3M)
pow, sqrt: exponential, logarithm, power, square root functions..... exp(3M)
pr: print files..... pr(1)
/dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct,/ ..... acctsh(1M)
/lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup,/ ..... acctsh(1M)
bc: arbitrary- precision arithmetic language..... bc(1)
atof: convert string to double- precision number ..... strtod(3C)
.environ, .pref, .variables: FACE files..... env(4)
monitor: prepare execution profile..... monitor(3C)
cpp: the C language preprocessor ..... see *C and Lang Tools Binder*
factor: obtain the prime factors of a number ..... factor(1)
gps: graphical primitive string, format of graphical files..... gps(4)
types: primitive system data types..... types(5)
prs: print an SCCS file..... prs(1)
date: print and set the date ..... date(1)
cal: print calendar ..... cal(1)

sum: print checksum and block count of a file.....sum(1)
sact: print current SCCS file editing activity.....sact(1)
cat: concatenate and print files.....cat(1)
pr: print files.....pr(1)
printf, fprintf, sprintf: print formatted output.....printf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs/.....vprintf(3S)
of the LP print service lpstat: print information about the status.....lpstat(1)
information about RCS files rlog: print log messages and other.....rlog(1)
object file nm: print name list of.....see *C and Lang Tools Binder*
uname: print name of current system.....uname(1)
news: print news items.....news(1)
infocmp: compare or print out terminfo descriptions.....infocmp(1M)
acctcom: search and print process accounting files.....acctcom(1)
of object files size: print section sizes.....see *C and Lang Tools Binder*
send/cancel requests to the LP print service.....lp(1)
lpadmin: configure the LP print service.....lpadmin(1M)
administer filters used with the LP print service.....lpfilter(1M)
administer forms used with the LP print service.....lpforms(1M)
about the status of the LP print service.....lpstat(1)
/lpshut, lpmove: start/stop the LP print service and move requests.....lpsched(1M)
strace: print STREAMS trace messages.....strace(1M)
prtvtoc: print the VTOC of a block device.....prtvtoc(1M)
prtvtoc: print the VTOC of a standalone disk device.....prtvtoc(8)
id: print user and group IDs and names.....id(1M)
lpmod: parallel printer canonical processing module.....lpmod(7)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
lptypes: table of printer types supported by the menu system.....lptypes(4)
enable, disable: enable/disable LP printers.....enable(1)
formatted output printf, fprintf, sprintf: print.....printf(3S)
lpusers: set printing queue priorities.....lpusers(1M)
lpusers: set printing queue priorities.....lpusers(1M)
nice: run a command at low priority.....nice(1)
/noswap, onproc: run a command with priority aging, PFF, swapping, or/.....noage(1M)
nice: change priority of a process.....nice(2)
brk, sbrk: change private data segment size.....brk(3PPS)
process attributes proc_ctl: control over various.....proc_ctl(2SEQ)
codestroy: communicate to a process.....coproc(1F)
exit, _exit: terminate process.....exit(2)
fork: create a new process.....fork(2)
inittab: script for the init process.....inittab(4)
kill: terminate a process.....kill(1)
nice: change priority of a process.....nice(2)
pclose: initiate pipe to/from a process.....popen(3S)
wait: await completion of process.....wait(1)
acct: enable or disable process accounting.....acct(2)
acctprc1, acctprc2: process accounting.....acctprc(1M)
acct: per- process accounting file format.....acct(4)
acctcom: search and print process accounting files.....acctcom(1)
acctcms: command summary from per- process accounting records.....acctcms(1M)
alarm: set a process alarm clock.....alarm(2)
times: get process and child process times.....times(2)

proc_ctl: control over various process attributes.....proc_ctl(2SEQ)
m_single, m_multi: mark single- process code section.....m_single(3PPS)
ipcs: report inter- process communication facilities status.....ipcs(1)
init, telinit: process control initialization.....init(1M)
timex: time a command; report process data and system activity.....timex(1)
suspend and resume child process execution.....m_park_procs(3PPS)
setpgrp: run command in its own process group.....setpgrp(1)
/getpgrp, getppid: get process, process group, and parent process IDs.....getpid(2)
setpgrp: set process group ID.....setpgrp(2)
setsid: create session and set process group ID.....setsid(2)
tcgetpgrp: get foreground process group ID.....tcgetpgrp(2)
tcsetpgrp: set foreground process group ID.....tcsetpgrp(2)
m_get_myid: return process identification.....m_get_myid(3PPS)
process, process group, and parent process IDs.....getpid(2)
vfork: spawn new process in a virtual memory efficient way.....vfork(2)
kill: send a signal to a process or a group of processes.....kill(2)
getpid, getpgrp, getppid: get process, process group, and parent/.....getpid(2)
ps: report process status.....ps(1)
plock: lock process, text, or data in memory.....plock(2)
times: get process and child process times.....times(2)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
wait, waitpid: wait for a child process to stop or terminate.....wait(2)
ptrace, mptrace: process trace facility.....ptrace(2)
pause: suspend process until signal.....pause(2)
vsig: synchronize a co- process with the controlling FMLI object.....vsig(1F)
checklist: list of filesystems processed by fsck and ncheck.....checklist(4)
a signal to a process or a group of processes.....kill(2)
killall: kill all active processes.....killall(1M)
return number of child processes.....m_get_numprocs(3PPS)
m_kill_procs: kill child processes.....m_kill_procs(3PPS)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
fuser: identify processes using a file or file structure.....fuser(1M)
m4: macro processor.....m4(1)
menu: menu processor.....menu(1M)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
tmp_ctl: multi- processor control.....tmp_ctl(2SEQ)
priority aging, PFF, swapping, or processor migration disabled.....noage(1M)
offline_all: take all but one active processor off line.....offline_all(3SEQ)
pdp11, u3b, u3b2, u3b5, vax: get processor type truth value.....machid(1)
online, offline: bring processors online and offline.....online(1M)
mmap: map an open file into the processs address space.....mmap(2SEQ)
munmap: unmap part of the processs address space.....munmap(2SEQ)
pmap: map physical addresses into processs virtual address space.....pmap(7)
product: installed products file.....product(4)
prof: display profile data.....prof(1)
prof: profile within a function.....prof(5)
profil: execution time profile.....profil(2)
monitor: prepare execution profile.....monitor(3C)
profil: execution time profile.....profil(2)
prof: display profile data.....prof(1)
at login time profile: setting up an environment.....profile(4)
prof: profile within a function.....prof(5)

end, etext, edata: last locations in program ..... end(3C)
mt: magnetic tape manipulating program ..... mt(1)
prompt: prompt program ..... prompt(1M)
sdiff: side-by-side difference program ..... sdiff(1)
STREAMS error logger cleanup program ..... strclean(1M)
units: conversion program ..... units(1)
for the uucp file transport program ..... uusched(1M)
assert: verify program assertion ..... assert(3X)
ASSIST menus or command/ astgen: program for generating/modifying ..... astgen(1)
uucico: file transport program for the uucp system ..... uucico(1M)
installboot: installs a boot program on a disk ..... installboot(1M)
boot: run a standalone program or bring up the operating system ..... boot(8)
a standard/restricted command/ programming language ..... ksh(1)
the standard/restricted command programming language ..... sh(1)
intro: introduction to Parallel Programming Library ..... intro(3PPS)
maintenance commands and application programs ..... intro(1)
update, and regenerate groups of programs ..... make(1)
lex: generate programs for simple lexical tasks ..... lex(1)
setlocale: set programs location ..... setlocale(3C)
prompt: prompt program ..... prompt(1M)
prompt: prompt program ..... prompt(1M)
proto: prototype job file for at ..... proto(4)
t_getinfo: get protocol-specific service information ..... t_getinfo(3N)
proto: prototype job file for at ..... proto(4)
prs: print an SCCS file ..... prs(1)
/monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup,/ ..... acctsh(1M)
device prtvtoc: print the VTOC of a block ..... prtvtoc(1M)
standalone disk device prtvtoc: print the VTOC of a ..... prtvtoc(8)
ps: report process status ..... ps(1)
streampty: STREAMS pseudo terminal driver ..... streampty(7)
getpseudotty: get pseudo terminals ..... getpseudotty(3SEQ)
cn: pseudoconsole driver ..... cn(7)
generate uniformly distributed pseudorandom numbers ..... drand48(3C)
time, ptime: time a command ..... time(1)
facility ptrace, mptrace: process trace ..... ptrace(2)
uuto, uupick: public UNIX-to-UNIX system file copy ..... uuto(1C)
ungetc: push character back into input stream ..... ungetc(3S)
puts, fputs: put a string on a stream ..... puts(3S)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
getdents: read directory entries and put in a file system independent/ ..... getdents(2)
echo: put string on virtual output ..... echo(1F)
character or word on a stream putc, putchar, fputc, putw: put ..... putc(3S)
or word on a stream putc, putchar, fputc, putw: put character ..... putc(3S)
environment putenv: change or add value to ..... putenv(3C)
putmsg: send a message on a stream ..... putmsg(2)
putpwent: write password file entry ..... putpwent(3C)
stream puts, fputs: put a string on a ..... puts(3S)
message: puts its arguments on message line ..... message(1F)
entry putspent: write shadow password file ..... putspent(3X)
getut: getutent, getutid, getutline, pututline, setutent, endutent,/ ..... getut(3C)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
checkers pwck, grpck: password/group file ..... pwck(1M)

/etc/shadow with information from/ pwconv: installs and updates ..... pwconv(1M)
pwd: working directory name ..... pwd(1)
database creation pwdhd: daemon to oversee password ..... pwdhd(1M)
password entries pwfix: remove incorrectly formatted ..... pwfix(1M)
one password file scheme pwunconv: converts from a two to a ..... pwunconv(1M)
qsort: quicker sort ..... qsort(3C)
tput: initialize a terminal or query terminfo database ..... tput(1)
route user message to menu message queue ..... menumsg(1M)
msgget: get message queue ..... msgget(2)
lpusers: set printing queue priorities ..... lpusers(1M)
ipcrm: remove a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
quotacheck: filesystem quota consistency checker ..... quotacheck(1M)
quota: display disk usage and limits ..... quota(1)
consistency checker quotacheck: filesystem quota ..... quotacheck(1M)
quotactl: manipulate disk quotas ..... quotactl(2SEQ)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
quotas on and off quotaon, quotaoff: turn filesystem ..... quotaon(1M)
edquota: edit user quotas ..... edquota(1M)
quotactl: manipulate disk quotas ..... quotactl(2SEQ)
repquota: summarize quotas for a filesystem ..... repquota(1M)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
generator rand, srand: simple random-number ..... rand(3C)
fsirand: install random inode generation numbers ..... fsirand(1M)
rand, srand: simple random-number generator ..... rand(3C)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
ratfor: rational Fortran dialect ..... ratfor(1)
ratfor: rational Fortran dialect ..... ratfor(1)
scripts rc: reboot and shutdown command ..... rc(4)
the operating system rc0: run commands performed to stop ..... rc0(1M)
multi-user environment rc2: run commands performed for ..... rc2(1M)
rcs: change RCS file attributes ..... rcs(1)
rcsintro: introduction to RCS commands ..... rcsintro(1)
rcsfile: format of RCS file ..... rcsfile(4)
rcs: change RCS file attributes ..... rcs(1)
sccstorcs: build RCS file from SCCS file ..... sccstorcs(1)
messages and other information about RCS files ..... rlog(1)
ci: check in RCS revisions ..... ci(1)
co: check out RCS revisions ..... co(1)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
rcsfile: format of RCS file ..... rcsfile(4)
commands rcsintro: introduction to RCS ..... rcsintro(1)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
getpass: read a password ..... getpass(3C)
of a common object file ldtbread: read an indexed symbol table entry ..... ldtbread(3X)
header of a/ ldshread, ldnshread: read an indexednamed section ..... ldshread(3X)
structure read_constab: read constab entries into a ..... read_constab(3SEQ)
file system independent/ getdents: read directory entries and put in a ..... getdents(2)
read: read from file ..... read(2)
mail, rmail: send mail to users or read mail ..... mail(1)
line: read one line ..... line(1)

read: read from file ..... read(2)
of an archive file ldahread: read the archive header of a member..... ldahread(3X)
object file ldfhread: read the file header of a common..... ldfhread(3X)
readlink: read value of a symbolic link..... readlink(2SEQ)
readv: read vector from file..... readv(2SEQ)
into a structure read_constab: read constab entries..... read_constab(3SEQ)
directory: opendir, readdir, telldir, seekdir, rewinddir,/..... directory(3X)
gets longest line readfile, longline: reads file and..... readfile(1F)
open a common object file for reading..... ldopen(3X)
open: open for reading or writing ..... open(2)
link readlink: read value of a symbolic..... readlink(2SEQ)
readfile, longline: reads file and gets longest line ..... readfile(1F)
readv: read vector from file..... readv(2SEQ)
lseek: move read/write file pointer ..... lseek(2)
tirdwr: Transport Interface read/write interface STREAMS module..... tirdwr(7)
/get real user, effective user, real group, and effective group IDs..... getuid(2)
/geteuid, getgid, getegid: get real user, effective user, real/..... getuid(2)
malloc, free, realloc, calloc: main memory allocator..... malloc(3C)
fast main memory/ malloc, free, realloc, calloc, mallopt, mallinfo:..... malloc(3X)
rc: reboot and shutdown command scripts..... rc(4)
t_rcvudata: receive a data unit..... t_rcvudata(3N)
t_rcvuderr: receive a unit data error indication ..... t_rcvuderr(3N)
over a connection t_rcv: receive data or expedited data sent..... t_rcv(3N)
connect request t_rcvconnect: receive the confirmation from a ..... t_rcvconnect(3N)
lockf: record locking on files..... lockf(3C)
ed, red: text editor ..... ed(1)
setcolor: redefine or create a color..... setcolor(1F)
dltsectbl: refresh security table..... dltsectbl(1M)
regular expression regcmp, regex: compile and execute..... regcmp(3X)
regcmp: regular expression compile ..... regcmp(1)
make: maintain, update, and regenerate groups of programs..... make(1)
expression regcmp, regex: compile and execute regular..... regcmp(3X)
string regex: match patterns against a ..... regex(1F)
and match routines regexp: regular expression compile..... regexp(5)
regcmp, regex: compile and execute regular expression..... regcmp(3X)
regcmp: regular expression compile..... regcmp(1)
routines regexp: regular expression compile and match..... regexp(5)
a file for a pattern using full regular expressions..... egrep(1)
reinit: run an initialization file..... reinit(1F)
accept, reject: allow or prevent LP requests..... accept(1M)
comm: select or reject lines common to two sorted files ..... comm(1)
join: relational database operator ..... join(1)
t_sndrel: initiate an orderly release..... t_sndrel(3N)
acknowledge receipt of an orderly release indication..... t_rcvrel(3N)
common object file reloc: relocation information for a..... reloc(4)
ldrseek, ldnrseek: seek to relocation entries of a section of a common/..... ldrseek(3X)
reloc: relocation information for a common object file ..... reloc(4)
/ceil, fmod, fabs: floor, ceiling, remainder, absolute value functions..... floor(3M)
calendar: reminder service..... calendar(1)
uuxqt: execute remote command requests..... uuxqt(1M)
Uutry: try to contact remote system with debugging on..... uutry(1M)
ct: spawn getty to a remote terminal..... ct(1C)

rmdel: remove a delta from an SCCS file.....rmdel(1)
rmdir: remove a directory.....rmdir(2)
rmvopt: remove a menu option.....rmvopt(1M)
set, or shared memory ID ipcrm: remove a message queue, semaphore.....ipcrm(1)
unlink: remove directory entry.....unlink(2)
rm, rmdir: remove files or directories.....rm(1)
password entries pwfix: remove incorrectly formatted.....pwfix(1M)
deroff: remove nroff, tbl, and eqn constructs.....deroff(1)
rename: change the name of a file.....rename(2)
consistency check and interactive repair.....fsck(1M)
uniq: report repeated lines in a file.....uniq(1)
clock: report CPU time used.....clock(3C)
fsstat: report filesystem status.....fsstat(1M)
facilities status ipcs: report inter-process communication.....ipcs(1)
df: report number of free disk blocks and inodes.....df(1M)
mailbug: submit a bug report or enhancement request.....mailbug(1)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
timex: time a command; report process data and system activity.....timex(1)
ps: report process status.....ps(1)
uniq: report repeated lines in a file.....uniq(1)
fseek, rewind, ftell: reposition a file pointer in a stream.....fseek(3S)
filesystem repquota: summarize quotas for a.....repquota(1M)
submit a bug report or enhancement request.....mailbug(1)
t_accept: accept a connect request.....t_accept(3N)
t_listen: listen for a connect request.....t_listen(3N)
the confirmation from a connect request.....t_rcvconnect(3N)
send user-initiated disconnect request.....t_snddis(3N)
accept, reject: allow or prevent LP requests.....accept(1M)
the LP print service and move requests.....lpsched(1M)
uuxqt: execute remote command requests.....uuxqt(1M)
lp, cancel: send/cancel requests to the LP print service.....lp(1)
to its default values reset: reset the current form field.....reset(1F)
default values reset: reset the current form field to its.....reset(1F)
get_process_stats: get resource utilization info.....get_process_stats(2SEQ)
restore: incremental filesystem restore.....restore(1M)
restore restore: incremental filesystem.....restore(1M)
/ksh, rksh – KornShell, a standard/ restricted command and programming/.....ksh(1)
sh, rsh: shell, the standard/ restricted command programming/.....sh(1)
shadow: restricted password file.....shadow(4)
/m_rele_procs: suspend and resume child process execution.....m_park_procs(3PPS)
usage examples usage: retrieve a command description and.....usage(1)
t_rcvdis: retrieve information from disconnect.....t_rcvdis(3N)
object file symbol table/ ldgetname: retrieve symbol name for common.....ldgetname(3X)
abs: return integer absolute value.....abs(3C)
logname: return login name of user.....logname(3X)
m_get_numprocs: return number of child procs.....m_get_numprocs(3PPS)
m_get_myid: return process identification.....m_get_myid(3PPS)
getenv: return value for environment name.....getenv(3C)
stat: data returned by stat system call.....stat(5)
marked menu items getitems: returns a list of the currently.....getitems(1F)
getfrm: returns the current frame number.....getfrm(1F)
cpus_online: returns the number of CPUs on-line.....cpus_online(3PPS)

col: filter reverse line-feeds.....col(1)
ci: check in RCS revisions.....ci(1)
co: check out RCS revisions.....co(1)
rcsdiff: compare RCS revisions.....rcsdiff(1)
rcsmerge: merge RCS revisions.....rcsmerge(1)
pointer in a stream fseek, rewind, ftell: reposition a file.....fseek(3S)
/opendir, readdir, telldir, seekdir, rewinddir, closedir: directory/.....directory(3X)
creat: create a new file or rewrite an existing one.....creat(2)
restricted command/ ksh, rksh: KornShell, a standard/.....ksh(1)
information about RCS files rlog: print log messages and other.....rlog(1)
directories rm, rmdir: remove files or.....rm(1)
mail, rmail: send mail to users or read mail.....mail(1)
file rmdel: remove a delta from an SCCS.....rmdel(1)
rmdir: remove a directory.....rmdir(2)
rm, rmdir: remove files or directories.....rm(1)
rmvopt: remove a menu option.....rmvopt(1M)
chroot: change root directory.....chroot(2)
chroot: change root directory for a command.....chroot(1M)
logarithm, power, square root functions.....exp(3M)
queue menumsg: route user message to menu message.....menumsg(1M)
ldfcn: common object file access routines.....ldfcn(4)
regular expression compile and match routines.....regexp(5)
programming language sh, rsh: shell, standard/restricted command/.....sh(1)
nice: run a command at low priority.....nice(1)
nohup: run a command immune to hangups and quits.....nohup(1)
shell: run a command using shell.....shell(1F)
noage, nopff, noswap, onproc: run a command with priority aging, PFF/.....noage(1M)
the operating system boot: run a standalone program or bring up.....boot(8)
run: run an executable.....run(1F)
edc: run an externally described command.....edc(1M)
reinit: run an initialization file.....reinit(1F)
setpgrp: run command in its own process group.....setpgrp(1)
multi-user environment rc2: run commands performed for.....rc2(1M)
operating system rc0: run commands performed to stop the.....rc0(1M)
runacct: run daily accounting.....runacct(1M)
run: run an executable.....run(1F)
runacct: run daily accounting.....runacct(1M)
Administration SA: devices administered by System.....sa(7)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
editing activity sact: print current SCCS file.....sact(1)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
report package sar: sa1, sa2, sadc: system activity.....sar(1M)
sar: system activity reporter.....sar(1)
savecore: save a core dump of the operating system.....savecore(1M)
operating system savecore: save a core dump of the.....savecore(1M)
brk, sbrk: change data segment space allocation.....brk(2)
brk, sbrk: change private data segment size.....brk(3PPS)
formatted input scanf, fscanf, sscanf: convert.....scanf(3S)
bfs: big file scanner.....bfs(1)
awk: pattern scanning and processing language.....awk(1)
nawk: pattern scanning and processing language.....nawk(1)

change the delta commentary of an SCCS delta ..... cdc(1)
comb: combine SCCS deltas ..... comb(1)
delta: make a delta (change) to an SCCS file ..... delta(1)
get: get a version of an SCCS file ..... get(1)
prs: print an SCCS file ..... prs(1)
rmdel: remove a delta from an SCCS file ..... rmdel(1)
sccsdiff: compare two versions of an SCCS file ..... sccsdiff(1)
sccsfile: format of SCCS file ..... sccsfile(4)
sccstorcs: build RCS file from SCCS file ..... sccstorcs(1)
unget: undo a previous get of an SCCS file ..... unget(1)
val: validate SCCS file ..... val(1)
sact: print current SCCS file editing activity ..... sact(1)
admin: create and administer SCCS files ..... admin(1)
what: identify SCCS files ..... what(1)
SCCS file sccsdiff: compare two versions of an ..... sccsdiff(1)
sccsfile: format of SCCS file ..... sccsfile(4)
file sccstorcs: build RCS file from SCCS ..... sccstorcs(1)
board SCED: SCSI/Ethernet/Diagnostics ..... sced(7)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
transport program uusched: the scheduler for the uucp file ..... uusched(1M)
unlock locks s_init_lock, s_lock, s_clock, s_unlock: initialize, lock, ... s_lock(3PPS)
object file scnhdr: section header for a common ..... scnhdr(4)
image file scr_dump: format of curses screen ..... scr_dump(4)
more: view a file one full screen at a time ..... more(1)
curses: terminal screen handling and optimization package ..... curses(3X)
scr_dump: format of curses screen image file ..... scr_dump(4)
vi: screen-oriented (visual) display editor based on ex ..... vi(1)
inittab: script for the init process ..... inittab(4)
rc: reboot and shutdown command scripts ..... rc(4)
tg: SCSI 1/2-inch magnetic tape interface ..... tg(7)
sd: SCSI disk controller ..... sd(7)
wd: SSM SCSI disk controller ..... wd(7)
CCSformat: standalone embedded SCSI disk formatter ..... ccsformat(8)
tm: SCSI tape controller ..... tm(7)
ts: SCSI tape controller ..... ts(7)
SCED: SCSI/Ethernet/Diagnostics board ..... sced(7)
getscsiinfo, getscsimatch: get scsiinfo structure ..... getscsiinfo(3SEQ)
sd: SCSI disk controller ..... sd(7)
program sdiff: side-by-side difference ..... sdiff(1)
fgrep: search a file for a character string ..... fgrep(1)
grep: search a file for a pattern ..... grep(1)
full regular expressions egrep: search a file for a pattern using ..... egrep(1)
bsearch: binary search a sorted table ..... bsearch(3C)
acctcom: search and print process accounting files ..... acctcom(1)
lsearch, lfind: linear search and update ..... lsearch(3C)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
hcreate, hdestroy: manage hash search tables ..... hsearch(3C)
tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)
manipulate the object file comment section ..... mcs(1)
m_multi: mark single-process code section ..... m_single(3PPS)
scnhdr: section header for a common object file ..... scnhdr(4)
/ldnshread: read an indexednamed section header of a common object/ ..... ldshread(3X)

/seek to line number entries of a — section of a common object file ..... ldlseek(3X)
/seek to relocation entries of a — section of a common object file ..... ldrseek(3X)
/ldnsseek: seek to an indexednamed — section of a common object file ..... ldsseek(3X)
dltsectbl: refresh — security table ..... dltsectbl(1M)
task: global — security table ..... task(4)
tasktbl: create — security table in shared memory ..... tasktbl(1M)
— sed: stream editor ..... sed(1)
/nrand48, mrand48, jrand48, srand48, — seed48, lcong48: generate uniformly/ ..... drand48(3C)
a common object/ ldsseek, ldnsseek: — seek to an indexednamed section of ..... ldsseek(3X)
section of a/ ldlseek, ldnlseek: — seek to line number entries of a ..... ldlseek(3X)
ldrseek, ldnrseek: — seek to relocation entries of a section of a/ ..... ldrseek(3X)
a common object file ldohseek: — seek to the optional file header of ..... ldohseek(3X)
object file ldtbseek: — seek to the symbol table of a common ..... ldtbseek(3X)
/opendir, readdir, telldir, — seekdir, rewinddir, closedir/ ..... directory(3X)
sweepoff: enable/disable patrol — seeks on DCC disks ..... sweepon(1M)
shmget: get shared memory — segment identifier ..... shmget(2)
brk, sbrk: change private data — segment size ..... brk(3PPS)
shbrk, shsbrk: change shared data — segment size ..... shbrk(3PPS)
brk, sbrk: change data — segment space allocation ..... brk(2)
sorted files comm: — select or reject lines common to two ..... comm(1)
greek: — select terminal filter ..... greek(1)
addformopt: add a form/ — SELECT window option to a menu ..... addformopt(1M)
semctl: — semaphore control operations ..... semctl(2)
semop: — semaphore operations ..... semop(2)
ipcrm: remove a message queue, — semaphore set, or shared memory ID ..... ipcrm(1)
semget: get set of — semaphores ..... semget(2)
— semctl: semaphore control operations ..... semctl(2)
— semget: get set of semaphores ..... semget(2)
— semop: semaphore operations ..... semop(2)
t_sndudata: — send a data unit ..... t_sndudata(3N)
putmsg: — send a message on a stream ..... putmsg(2)
kill: — send a signal to a process or a group of processes ..... kill(2)
tfnotify: — send completion message ..... tfnotify(1M)
connection t_snd: — send data or expedited data over a ..... t_snd(3N)
mail, rmail: — send mail to users or read mail ..... mail(1)
t_snddis: — send user-initiated disconnect request ..... t_snddis(3N)
lp, cancel: — send/cancel requests to the LP print service ..... lp(1)
receive data or expedited data — sent over a connection ..... t_rcv(3N)
intro: introduction to — Sequent system calls ..... intro(2SEQ)
dis386: disassemble — Series 386 instructions ..... dis386(3SEQ)
SSM: System — Services Module ..... ssm(7)
setsid: create — session and set process group ID ..... setsid(2)
ascii: map of ASCII character — set ..... ascii(5)
alarm: — set a process alarm clock ..... alarm(2)
umask: — set and get file creation mask ..... umask(2)
stty, gtty: — set and get terminal state (defunct) ..... stty(2)
in core or in files set, unset: — set and unset environment variables ..... set(1F)
sigstack: — set and/or get signal stack context ..... sigstack(2SEQ)
timezone: — set default system time zone ..... timezone(4)
backup.info: backup — set definitions ..... backup_info(4)
setdtablesize: — set descriptor table size ..... setdtablesize(2SEQ)
env: — set environment for command execution ..... env(1)

utime: set file access and modification times.....utime(2)
umask: set file-creation mode mask.....umask(1)
tcsetpgrp: set foreground process group ID.....tcsetpgrp(2)
setgroups: set group access list.....setgroups(2SEQ)
cpset: copy file and set mode, user, and group IDs.....cpset(1)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
semget: get set of semaphores.....semget(2)
remove a message queue, semaphore set, or shared memory ID.....ipcrm(1)
lpusers: set printing queue priorities.....lpusers(1M)
setpgrp: set process group ID.....setpgrp(2)
setsid: create session and set process group ID.....setsid(2)
setlocale: set programs location.....setlocale(3C)
tabs: set tabs on a terminal.....tabs(1)
tcgetattr, tcsetattr: get/ set terminal state.....tcgetattr(2)
line discipline getty: set terminal type, modes, speed, and.....getty(1M)
line discipline uugetty: set terminal type, modes, speed, and.....uugetty(1M)
date: print and set the date.....date(1)
stty: set the options for a terminal.....stty(1)
stime: set time.....stime(2)
environment variables in core or in/ set, unset: set and unset.....set(1F)
setuid, setgid: set user and group IDs.....setuid(2)
ulimit: get and set user limits.....ulimit(2)
a stream setbuf, setvbuf: assign buffering to.....setbuf(3S)
/toascci, _tolower, _toupper, setchrclass: character handling.....ctype(3C)
setcolor: redefine or create a color.....setcolor(1F)
size setdtablesize: set descriptor table.....setdtablesize(2SEQ)
setuid, setgid: set user and group IDs.....setuid(2)
getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent: get group/.....getgrent(3C)
setgroups: set group access list.....setgroups(2SEQ)
setjmp, longjmp: non-local goto.....setjmp(3C)
crypt, setkey, encrypt: generate hashing encryption.....crypt(3C)
setlocale: set programs location.....setlocale(3C)
setmnt: establish mount table.....setmnt(1M)
/endmntent, getmntent, hasmntopt, setmntent: get file system/.....getmntent(3SEQ)
setpgid: manipulate signal sets.....setpgid(2)
process group setpgrp: run command in its own.....setpgrp(1)
setpgrp: set process group ID.....setpgrp(2)
getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get/.....getpwent(3C)
process group ID setsid: create session and set.....setsid(2)
getspent, getspnam, setspent, endspent, fgetspent, lckpwdf/.....getspent(3X)
profile: setting up an environment at login time.....profile(4)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
IDs setuid, setgid: set user and group.....setuid(2)
user setup: initialize system for first.....setup(1)
/getutid, getutline, pututline, setutent, endutent, utmpname/.....getut(3C)
setbuf, setvbuf: assign buffering to a stream.....setbuf(3S)
machine-independent fashion sputl, sgetl: access long integer data in a.....sputl(3X)
standard/restricted command/ sh, rsh: shell, the.....sh(1)
/fgetspent, lckpwdf, ulckpwdf: get shadow password file entry.....getspent(3X)
putspent: write shadow password file entry.....putspent(3X)
shadow: restricted password file.....shadow(4)
pwconv: installs and updates /etc/ shadow with information from/.....pwconv(1M)

shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
tasktbl: create security table in shared memory ..... tasktbl(1M)
shmalloc, shrealloc, shfree: shared memory allocator ..... shmalloc(3PPS)
shmctl: shared memory control operations ..... shmctl(2)
a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
shmget: get shared memory segment identifier ..... shmget(2)
segment size shbrk, shsbrk: change shared data ..... shbrk(3PPS)
shell: run a command using shell ..... shell(1F)
assign menu system as user login shell ..... tfmenush(1M)
system: issue a shell command ..... system(3S)
csh: a shell (command interpreter) with C-like syntax ..... csh(1)
/shutacct, startup, turnacct: shell procedures for accounting ..... acctsh(1M)
shell: run a command using shell ..... shell(1F)
shells: shell table ..... shells(4)
command programming/ sh, rsh: shell, the standard/restricted ..... sh(1)
shells: shell table ..... shells(4)
shmalloc, shrealloc, shfree: shared memory allocator ..... shmalloc(3PPS)
memory allocator shmalloc, shrealloc, shfree: shared ..... shmalloc(3PPS)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
operations shmctl: shared memory control ..... shmctl(2)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
identifier shmget: get shared memory segment ..... shmget(2)
operations shmop, shmat, shmdt: shared memory ..... shmop(2)
the machine showcfg: show the configuration of ..... showcfg(1M)
allocator shmalloc, shrealloc, shfree: shared memory ..... shmalloc(3PPS)
shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
shutdown: shut down system, change system state ..... shutdown(1M)
/nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct: shell/ ..... acctsh(1M)
rc: reboot and shutdown command scripts ..... rc(4)
shutdown: system shutdown information file ..... shutdown(4)
system state shutdown: shut down system, change ..... shutdown(1M)
information file shutdown: system shutdown ..... shutdown(4)
actions sigaction: examine and change signal ..... sigaction(2)
manipulate/ sigemptyset, sigfillset, sigaddset, sigdelset, sigismember: ..... sigemptyset(2)
sigcontext: get signal context ..... sigcontext(2SEQ)
sigemptyset, sigfillset, sigaddset, sigdelset, sigismember: manipulate/ ..... sigemptyset(2)
sigdelset, sigismember: manipulate/ sigemptyset, sigfillset, sigaddset, ... sigemptyset(2)
sigismember/ sigemptyset, sigfillset, sigaddset, sigdelset, ... sigemptyset(2)
sigpause: signal management sigset, sighold, sigrelse, sigignore, ... sigset(2)
sigset, sighold, sigrelse, sigignore, sigpause: signal/ ..... sigset(2)
/sigfillset, sigaddset, sigdelset, sigismember: manipulate signal sets ..... sigemptyset(2)
sigsetjmp, siglongjmp: non-local jumps ..... sigsetjmp(3C)
login: sign on ..... login(1)
pause: suspend process until signal ..... pause(2)
specify what to do upon receipt of a signal ..... signal(2)
sigsuspend: wait for a signal ..... sigsuspend(2)
sigaction: examine and change signal actions ..... sigaction(2)
sigcontext: get signal context ..... sigcontext(2SEQ)
sigrelse, sigignore, sigpause: signal management ..... sigset(2)
setpgid: manipulate signal sets ..... setpgid(2)
sigdelset, sigismember: manipulate signal sets ..... sigemptyset(2)

receipt of a signal signal: specify what to do upon.....signal(2)
sigstack: set and/or get signal stack context.....sigstack(2SEQ)
kill: send a signal to a process or a group of processes.....kill(2)
sigpending: examine pending signals.....sigpending(2)
examine and change blocked signals.....sigprocmask(2)
ssignal, gsignal: software signals.....ssignal(3C)
sighold, sigrelse, sigignore, sigpause: signal management.....sigset(2)
sigpending: examine pending signals.....sigpending(2)
blocked signals sigprocmask: examine and change.....sigprocmask(2)
signal management sigset, sighold, sigrelse, sigignore, sigpause:.....sigset(2)
sigignore, sigpause: signal/ sigset, sighold, sigrelse,.....sigset(2)
jumps sigsetjmp, siglongjmp: non-local.....sigsetjmp(3C)
stack context sigstack: set and/or get signal.....sigstack(2SEQ)
sigsuspend: wait for a signal.....sigsuspend(2)
atan2: trigonometric/ trig: sin, cos, tan, asin, acos, atan,.....trig(3M)
m_single, m_multi: mark single-process code section.....m_single(3PPS)
functions sinh, cosh, tanh: hyperbolic.....sinh(3M)
initialize barrier, wait at barrier s_init_barrier, s_wait_barrier:.....s_wait_barrier(3PPS)
s_unlock: initialize, lock, unlock/ s_init_lock, s_lock, s_clock,.....s_lock(3PPS)
sbrk: change private data segment size.....brk(3PPS)
getdtablesize: get descriptor table size.....getdtablesize(2SEQ)
getpagesize: get system page size.....getpagesize(2SEQ)
setdtablesize: set descriptor table size.....setdtablesize(2SEQ)
shsbrk: change shared data segment size.....shbrk(3PPS)
of object files size: print section sizes.....see *C and Lang Tools Binder*
interval sleep: suspend execution for an.....sleep(1)
interval sleep: suspend execution for.....sleep(3C)
s_init_lock, s_lock, s_clock, s_unlock: initialize, lock,/.....s_lock(3PPS)
ttyslot: find the slot in the utmp file of the current user.....ttyslot(3C)
consmem, sm, ss: console memory drivers.....consmem(7)
ssignal, gsignal: software signals.....ssignal(3C)
qsort: quicker sort.....qsort(3C)
tsort: topological sort.....tsort(1)
sort: sort and/or merge files.....sort(1)
sort: sort and/or merge files.....sort(1)
select or reject lines common to two sorted files.....comm(1)
bsearch: binary search a sorted table.....bsearch(3C)
an object file list: produce C source listing from.....see *C and Lang Tools Binder*
sp: STREAMS pipe device.....sp(7)
open file into the processs address space.....mmap(2SEQ)
unmap part of the processs address space.....munmap(2SEQ)
into processs virtual address space.....pmap(7)
brk, sbrk: change data segment space allocation.....brk(2)
ct: spawn getty to a remote terminal.....ct(1C)
memory efficient way vfork: spawn new process in a virtual.....vfork(2)
fspec: format specification in text files.....fspec(4)
signal: specify what to do upon receipt of a signal.....signal(2)
getty: set terminal type, modes, speed, and line discipline.....getty(1M)
uugetty: set terminal type, modes, speed, and line discipline.....uugetty(1M)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
find spelling errors spell, hashmake, spellin, hashcheck:.....spell(1)
file spellhist: unmatched spelling words.....spellhist(4)

spell, hashmake, hashmake, spellin, hashcheck: find — spellin, hashcheck: find spelling errors ..... spell(1)
spellhist: unmatched — spelling errors ..... spell(1)
csplit: context — spelling words file ..... spellhist(4)
split: — split ..... csplit(1)
fsplit: — split a file into pieces ..... split(1)
split: split a file into pieces ..... split(1)
uucleanup: uucp — spool directory clean-up ..... uucleanup(1M)
printf, fprintf, — sprintf: print formatted output ..... printf(3S)
data in a machine-independent/ — sputl, sgetl: access long integer ..... sputl(3X)
square root/ exp, log, log10, pow, — sqrt: exponential, logarithm, power, ... exp(3M)
sqrt: exponential, logarithm, power, — square root functions ..... exp(3M)
rand, — srand: simple random-number generator ..... rand(3C)
/lrand48, nrand48, mrand48, jrand48, — srand48, seed48, lcong48: generate/ ..... drand48(3C)
consmem, sm, — ss: console memory drivers ..... consmem(7)
scanf, fscanf, — sscanf: convert formatted input ..... scanf(3S)
ssignal, gsignal: software signals ..... ssignal(3C)
wp: — SSM parallel line printer interface ..... wp(7)
wd: — SSM SCSI disk controller ..... wd(7)
SSM: System Services Module ..... ssm(7)
terminal multiplexor driver — st: Systech MTI-800/1600/1650 ..... st(7)
sigstack: set and/or get signal — stack context ..... sigstack(2SEQ)
copy: perform a — standalone data transfer ..... copy(8)
prtvtoc: print the VTOC of a — standalone disk device ..... prtvtoc(8)
dual-channel disk/ zdformat: — standalone disk formatter for ..... zdformat(8)
CCSformat: — standalone embedded SCSI disk formatter ..... ccsformat(8)
dump: — standalone memory dump ..... dump(8)
operating system boot: run a — standalone program or bring up the ..... boot(8)
intro: introduction to system — standalone utilities ..... intro(8)
stdio: — standard buffered input/output package ..... stdio(3S)
package stdipc: ftok: — standard interprocess communication ..... stdipc(3C)
sh, rsh: shell, the — standard/restricted command programming/ ..... sh(1)
operating system for beginning/ — starter: information about the ..... starter(1)
lpsched, lpshut, lpmove: — start/stop the LP print service and move/ ..... lpsched(1M)
/prctmp, prdaily, prtacct, shutacct, — startup, turnacct: shell procedures/ ..... acctsh(1M)
call — stat: data returned by stat system ..... stat(5)
stat, lstat, fstat: get file status ..... stat(2)
stat: data returned by — stat system call ..... stat(5)
information — statfs, fstatfs: get filesystem ..... statfs(2)
ustat: get filesystem — statistics ..... ustat(2)
ff: list filenames and — statistics for a filesystem ..... ff(1M)
fsstat: report filesystem — status ..... fsstat(1M)
communication facilities — status ..... ipcs(1)
ps: report process — status ..... ps(1)
stat, lstat, fstat: get file — status ..... stat(2)
feof, clearerr, fileno: stream — status inquiries ..... ferror(3S)
uustat: uucp — status inquiry and job control ..... uustat(1C)
lpstat: print information about the — status of the LP print service ..... lpstat(1)
input/output package — stdio: standard buffered ..... stdio(3S)
communication package — stdipc: ftok: standard interprocess ..... stdipc(3C)
stime: set time ..... stime(2)
waitpid: wait for a child process to — stop or terminate ..... wait(2)

lpsched, lpshut, lpmove: start/ stop the LP print service and move/ ..... lpsched(1M)
rc0: run commands performed to stop the operating system ..... rc0(1M)
icheck: filesystem storage consistency check ..... icheck(1M)
strace: print STREAMS trace messages ..... strace(1M)
strncmp, strcpy, strncpy,/ string: strcat, strdup, strncat, strcmp, ... string(3C)
/strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn,/ ..... string(3C)
cleanup program strclean: STREAMS error logger ..... strclean(1M)
string: strcat, strdup, strncat, strcmp, strncmp, strcpy, strncpy,/ ..... string(3C)
/strdup, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr,/ ..... string(3C)
/strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations ..... string(3C)
strcpy, strncpy,/ string: strcat, strdup, strncat, strcmp, strncmp, ... string(3C)
fclose, fflush: close or flush a stream ..... fclose(3S)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
reposition a file pointer in a stream ..... fseek(3S)
getw: get character or word from a stream ..... getc(3S)
getmsg: get next message off a stream ..... getmsg(2)
gets, fgets: get a string from a stream ..... gets(3S)
putw: put character or word on a stream ..... putc(3S)
putmsg: send a message on a stream ..... putmsg(2)
puts, fputs: put a string on a stream ..... puts(3S)
setvbuf: assign buffering to a stream ..... setbuf(3S)
push character back into input stream ..... ungetc(3S)
sed: stream editor ..... sed(1)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
streamio: STREAMS ioctl commands ..... streamio(7)
driver streampty: STREAMS pseudo terminal ..... streampty(7)
clone: open any minor device on a STREAMS driver ..... clone(7)
strclean: STREAMS error logger cleanup program ..... strclean(1M)
strerr: STREAMS error logger daemon ..... strerr(1M)
log: interface to STREAMS error logging and event tracing ..... log(7)
poll: STREAMS input/output multiplexing ..... poll(2)
streamio: STREAMS ioctl commands ..... streamio(7)
Transport Interface cooperating STREAMS module ..... timod(7)
Interface read/write interface STREAMS module ..... tirdwr(7)
sp: STREAMS pipe device ..... sp(7)
streampty: STREAMS pseudo terminal driver ..... streampty(7)
strace: print STREAMS trace messages ..... strace(1M)
streamtty: Generic STREAMS tty driver interface ..... streamtty(7)
ld0: STREAMS tty line discipline ..... ld0(7)
driver interface streamtty: Generic STREAMS tty ..... streamtty(7)
strerr: STREAMS error logger daemon ..... strerr(1M)
strftime: format time information ..... strftime(3C)
long integer and base-64 ASCII string ..... a64l(3C)
tzset: convert date and time to string ..... ctime(3C)
convert floating-point number to string ..... ecvt(3C)
fgrep: search a file for a character string ..... fgrep(1)
regex: match patterns against a string ..... regex(1F)
gps: graphical primitive string, format of graphical files ..... gps(4)
gets, fgets: get a string from a stream ..... gets(3S)
puts, fputs: put a string on a stream ..... puts(3S)
echo: put string on virtual output ..... echo(1F)
strpbrk, strspn, strcspn, strtok: string operations ..... string(3C)

strcmp, strncmp, strcpy, strncpy,/ string: strcat, strdup, strncat, ...string(3C)
strtod, atof: convert string to double-precision number.....strtod(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
cftime: language specific strings .....cftime(4)
object file strip: strip an.....see *C and Lang Tools Binder*
/strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk,/.....string(3C)
string: strcat, strdup, strncat, strcmp, strncmp, strcpy, strncpy,/.....string(3C)
/strcat, strdup, strncat, strcmp, strncmp, strcpy, strlen,/.....string(3C)
/strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr,/.....string(3C)
/strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok:/.....string(3C)
/strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn,/.....string(3C)
/strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/.....string(3C)
character strstr: find first occurrence of.....strstr(3C)
double-precision number strtod, atof: convert string to.....strtod(3C)
strrchr, strpbrk, strspn, strcspn, strtok: string operations .....string(3C)
to integer strtol, atol, atoi: convert string .....strtol(3C)
processes using a file or file structure.....fuser(1M)
getscsimatch: get scsiinfo structure.....getscsiinfo(3SEQ)
getzdinfobyname: get zdinfo structure .....getzdinfo(3SEQ)
read constab entries into a structure .....read_constab(3SEQ)
t_alloc: allocate a library structure.....t_alloc(3N)
t_free: free a library structure.....t_free(3N)
state (defunct) stty, gtty: set and get terminal.....stty(2)
stty: set the options for a terminal.....stty(1)
su: become superuser or another user.....su(1M)
backup.codes1: backup subcode definitions.....backup_codes1(4)
m_fork: execute a subprogram in parallel.....m_fork(3PPS)
dbm_clearerr, dmb_blkflush: database subroutines .....ndbm(3X)
plot: graphics interface subroutines.....plot(3X)
sulog: log of substitute user events.....sulog(4)
sulog: log of substitute user events.....sulog(4)
of a file sum: print checksum and block count.....sum(1)
du: summarize disk usage.....du(1M)
repquota: summarize quotas for a filesystem .....repquota(1M)
records acctcms: command summary from per-process accounting .....acctcms(1M)
s_init_lock, s_lock, s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
sync: update the superblock .....sync(1M)
sync: update superblock.....sync(2)
su: become superuser or another user .....su(1M)
getgroups: get supplementary group IDs.....getgroups(2)
uucp: networking support files.....uucp(4)
lptypes: table of printer types supported by the menu system.....lptypes(4)
m_park_procs, m_rele_procs: suspend and resume child process/.....m_park_procs(3PPS)
sleep: suspend execution for an interval.....sleep(1)
sleep: suspend execution for interval.....sleep(3C)
pause: suspend process until signal.....pause(2)
swab: swap bytes.....swab(3C)
wait at barrier s_init_barrier, s_wait_barrier: initialize barrier,.....s_wait_barrier(3PPS)
swap: swap administrative interface.....swap(1M)
swab: swap bytes.....swab(3C)
swap: swap administrative interface .....swap(1M)
/a command with priority aging, PFF, swapping, or processor migration/.....noage(1M)

seeks on DCC disks sweepon, — sweepoff: enable/disable patrol ..... sweepon(1M)
patrol seeks on DCC disks — sweepon, sweepoff: enable/disable ..... sweepon(1M)
symbol table/ ldgetname: retrieve — symbol name for common object file ..... ldgetname(3X)
symbol name for common object file — symbol table entry ..... ldgetname(3X)
ldtbindex: compute the index of a — symbol table entry of a common/ ..... ldtbindex(3X)
ldtbread: read an indexed — symbol table entry of a common object/ ..... ldtbread(3X)
syms: common object file — symbol table format ..... syms(4)
ldtbseek: seek to the — symbol table of a common object file ..... ldtbseek(3X)
unistd: file header for — symbolic constants ..... unistd(4)
readlink: read value of a — symbolic link ..... readlink(2SEQ)
symlink: make — symbolic link to a file ..... symlink(2SEQ)
of common operating system terms and — symbols ..... glossary(1)
file — symlink: make symbolic link to a ..... symlink(2SEQ)
table format — syms: common object file symbol ..... syms(4)
sync: update superblock ..... sync(2)
sync: update the superblock ..... sync(1M)
controlling FMLI object vsig: — synchronize a co-process with the ..... vsig(1F)
t_sync: — synchronize transport library ..... t_sync(3N)
check documents for correct eqn — syntax ..... checkeq(1)
(command interpreter) with C-like — syntax ..... csh(1)
administration — sysadm: menu interface to do system ..... sysadm(1)
syscall: indirect system call ..... syscall(2)
variables — sysconf: get configurable system ..... sysconf(2)
configuration file — sys_config: format of a system ..... sys_config(4)
perror, errno, — sys_errlist, sys_nerr: system error messages ..... perror(3C)
sysi86: machine specific functions ..... sysi86(2)
menu system functions — syskeymap: identify keys to internal ..... syskeymap(4)
perror, errno, sys_errlist, — sys_nerr: system error messages ..... perror(3C)
multiplexor driver st: — Systech MTI-800/1600/1650 terminal ..... st(7)
lp: — Systech parallel line printer interface ..... lp(7)
bsearch: binary search a sorted — table ..... bsearch(3C)
constab: constructed devices — table ..... constab(4)
devtab: virtual device — table ..... devtab(4)
dltsectbl: refresh security — table ..... dltsectbl(1M)
kernel auto-configuration device — table ..... dumpconf(1M)
fstab: file-system- — table ..... fstab(4)
mnttab: mounted filesystem — table ..... mnttab(4)
setmnt: establish mount — table ..... setmnt(1M)
shells: shell — table ..... shells(4)
task: global security — table ..... task(4)
name for common object file symbol — table entry ..... ldgetname(3X)
/compute the index of a symbol — table entry of a common object file ..... ldtbindex(3X)
ldtbread: read an indexed symbol — table entry of a common object file ..... ldtbread(3X)
syms: common object file symbol — table format ..... syms(4)
tasktbl: create security — table in shared memory ..... tasktbl(1M)
get maximum in-use descriptor — table index ..... getdtablemax(2SEQ)
ldtbseek: seek to the symbol — table of a common object file ..... ldtbseek(3X)
the menu system lptypes: — table of printer types supported by ..... lptypes(4)
getdtablesize: get descriptor — table size ..... getdtablesize(2SEQ)
setdtablesize: set descriptor — table size ..... setdtablesize(2SEQ)
classification and conversion — tables ..... chrtbl(1M)
hdestroy: manage hash search — tables ..... hsearch(3C)

tabs: set — tabs on a terminal ..... tabs(1)
tabs: set tabs on a terminal ..... tabs(1)
t_accept: accept a connect request ..... t_accept(3N)
ctags: create a — tags file ..... ctags(1)
file — tail: deliver the last part of a ..... tail(1)
structure — t_alloc: allocate a library ..... t_alloc(3N)
trig: sin, cos, — tan, asin, acos, atan, atan2: trigonometric/ ..... trig(3M)
sinh, cosh, — tanh: hyperbolic functions ..... sinh(3M)
tm: SCSI — tape controller ..... tm(7)
ts: SCSI — tape controller ..... ts(7)
hpio: Hewlett-Packard 2645A terminal — tape file archiver ..... hpio(1)
tar: — tape file archiver ..... tar(1)
mtio: UNIX magnetic — tape interface ..... mtio(7)
tg: SCSI 1/2-inch magnetic — tape interface ..... tg(7)
xt: Xylogics 472 MULITBUS magnetic — tape interface ..... xt(7)
mt: magnetic — tape manipulating program ..... mt(1)
tar: tape file archiver ..... tar(1)
task: global security table ..... task(4)
shared memory — tasktbl: create security table in ..... tasktbl(1M)
transport endpoint — t_bind: bind an address to a ..... t_bind(3N)
deroff: remove nroff, — tbl, and eqn constructs ..... deroff(1)
terminal state — tcgetattr, tcsetattr: get/set ..... tcgetattr(2)
group ID — tcgetpgrp: get foreground process ..... tcgetpgrp(2)
t_close: close a transport endpoint ..... t_close(3N)
with another transport user — t_connect: establish a connection ..... t_connect(3N)
tcgetattr, — tcsetattr: get/set terminal state ..... tcgetattr(2)
group ID — tcsetpgrp: set foreground process ..... tcsetpgrp(2)
tsearch, tfind, — tdelete, twalk: manage binary search trees ..... tsearch(3C)
tee: pipe fitting ..... tee(1)
4014: paginator for the — Tektronix 4014 terminal ..... 4014(1)
init, — telinit: process control initialization ..... init(1M)
directory: opendir, readdir, — telldir, seekdir, rewinddir,/ ..... directory(3X)
temporary file tmpnam, — tempnam: create a name for a ..... tmpnam(3S)
tmpfile: create a — temporary file ..... tmpfile(3S)
tmpnam, tempnam: create a name for a — temporary file ..... tmpnam(3S)
terminals — term: conventional names for ..... term(5)
term: format of compiled — term file ..... term(4)
term: format of compiled term file ..... term(4)
description captoinfo: convert a — termcap description into a terminfo ..... captoinfo(1M)
base — termcap: terminal capability data ..... termcap(4)
paginator for the Tektronix 4014 — terminal ..... 4014(1)
special functions of the DASI 450 — terminal ..... 450(1)
ct: spawn getty to a remote — terminal ..... ct(1C)
ctermid: generate filename for — terminal ..... ctermid(3S)
hangup the specified controlling — terminal ..... fvhangup(2SEQ)
stty: set the options for a — terminal ..... stty(1)
tabs: set tabs on a — terminal ..... tabs(1)
tty: get the name of the — terminal ..... tty(1)
ttyname, isatty: find name of a — terminal ..... ttyname(3C)
/associated with a terminal or the — terminal associated with a device ..... ttydevname(1)
termcap: — terminal capability data base ..... termcap(4)
terminfo: — terminal capability database ..... terminfo(4)

streampty: STREAMS pseudo terminal driver.....streampty(7)
greek: select terminal filter.....greek(1)
tty: controlling terminal (gentty) interface.....tty(7)
termio: general terminal interface.....termio(7)
dial: establish an out-going terminal line connection.....dial(3C)
st: Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
tput: initialize a terminal or query terminfo database.....tput(1)
/get the device associated with a terminal or the terminal associated/.....ttydevname(1)
optimization package curses: terminal screen handling and.....curses(3X)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
tcgetattr, tcsetattr: get/set terminal state.....tcgetattr(2)
stty, gtty: set and get terminal state (defunct).....stty(2)
hpio: Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
line discipline getty: set terminal type, modes, speed, and.....getty(1M)
line discipline uugetty: set terminal type, modes, speed, and.....uugetty(1M)
functions of DASI 300 and 300s terminals.....300(1)
getpseudotty: get pseudo terminals.....getpseudotty(3SEQ)
special functions of Hewlett-Packard terminals.....hp(1)
term: conventional names for terminals.....term(5)
wait for a child process to stop or terminate.....wait(2)
kill: terminate a process.....kill(1)
exit, _exit: terminate process.....exit(2)
tic: terminfo compiler.....tic(1M)
tput: initialize a terminal or query terminfo database.....tput(1)
convert a termcap description into a terminfo description.....captoinfo(1M)
infocmp: compare or print out terminfo descriptions.....infocmp(1M)
database terminfo: terminal capability.....terminfo(4)
termio: general terminal interface.....termio(7)
of common operating system terms and symbols.....glossary(1)
t_error: produce error message.....t_error(3N)
test: condition evaluation command.....test(1)
isnan: isnand, isnanf: test for floating-point NaN/.....isnan(3C)
ed, red: text editor.....ed(1)
ex: text editor.....ex(1)
edit: text editor (variant of ex for casual users).....edit(1)
errormsg: error message text file.....errormsg(4)
newform: change the format of a text file.....newform(1)
fspec: format specification in text files.....fspec(4)
isismgr: ISIS file manager, updates text files in ISIS file format.....isismgr(1M)
plock: lock process, text, or data in memory.....plock(2)
search trees tsearch, tfind, tdelete, twalk: manage binary.....tsearch(3C)
reporter tfipc: interprocess communication.....tfipc(1M)
login shell tfmenush: assign menu system as user.....tfmenush(1M)
tfnotify: send completion message.....tfnotify(1M)
t_free: free a library structure.....t_free(3N)
interface tg: SCSI 1/2-inch magnetic tape.....tg(7)
service information t_getinfo: get protocol-specific.....t_getinfo(3N)
t_getstate: get the current state.....t_getstate(3N)
merge: three-way file merge.....merge(1)
tic: terminfo compiler.....tic(1M)
batch: execute commands at a later time.....at(1)

view a file one full screen at a time ..... more(1)
setting up an environment at login time ..... profile(4)
stime: set time ..... stime(2)
time: get time ..... time(2)
time, ptime: time a command ..... time(1)
and system activity timex: time a command; report process data ..... timex(1)
acctcon1, acctcon2: connect- time accounting ..... acctcon(1M)
time: get time ..... time(2)
strftime: format time information ..... strftime(3C)
profil: execution time profile ..... profil(2)
time, ptime: time a command ..... time(1)
mktime, tzset: convert date and time to string ..... ctime(3C)
clock: report CPU time used ..... clock(3C)
timezone: set default system time zone ..... timezone(4)
times: get process and child process times ..... times(2)
set file access and modification times ..... utime(2)
times: get process and child process ..... times(2)
update access and modification times of a file ..... touch(1)
process data and system activity timex: time a command; report ..... timex(1)
zone timezone: set default system time ..... timezone(4)
cooperating STREAMS module timod: Transport Interface ..... timod(7)
read/write interface STREAMS module tirdwr: Transport Interface ..... tirdwr(7)
request t_listen: listen for a connect ..... t_listen(3N)
a transport endpoint t_look: look at the current event on ..... t_look(3N)
tm: SCSI tape controller ..... tm(7)
processor tmp_affinity: bind a process to a ..... tmp_affinity(2SEQ)
tmp_ctl: multi-processor control ..... tmp_ctl(2SEQ)
tmpfile: create a temporary file ..... tmpfile(3S)
temporary file tmpnam, tempnam: create a name for a ..... tmpnam(3S)
/isgraph, isascii, tolower, toupper, toascci, _tolower, _toupper/ ..... ctype(3C)
/tolower, _toupper, _tolower, toascii: translate characters ..... conv(3C)
conv: toupper, tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
/isascii, tolower, toupper, toascci, _tolower, _toupper, setchrclass:/ ..... ctype(3C)
/ispunct, isprint, isgraph, isascii, tolower, toupper, toascci, _tolower,/ ..... ctype(3C)
conv: toupper, tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
endpoint t_open: establish a transport ..... t_open(3N)
tsort: topological sort ..... tsort(1)
transport endpoint t_optmgmt: manage options for a ..... t_optmgmt(3N)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
modification times of a file touch: update access and ..... touch(1)
/tolower, toupper, toascci, _tolower, _toupper, setchrclass: character/ ..... ctype(3C)
/isprint, isgraph, isascii, tolower, toupper, toascci, _tolower,/ ..... ctype(3C)
conv: toupper, tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
_tolower, toascii: translate/ conv: toupper, tolower, _toupper, ... conv(3C)
terminfo database tput: initialize a terminal or query ..... tput(1)
tr: translate characters ..... tr(1)
ptrace, mptrace: process trace facility ..... ptrace(2)
strace: print STREAMS trace messages ..... strace(1M)
to STREAMS error logging and event tracing ..... log(7)
copy: perform a standalone data transfer ..... copy(8)
init.d: system level transition files directory ..... init.d(4)
_toupper, _tolower, toascii: translate characters ..... conv(3C)

tr: translate characters ..... tr(1)
t_bind: bind an address to a transport endpoint ..... t_bind(3N)
t_close: close a transport endpoint ..... t_close(3N)
look at the current event on a transport endpoint ..... t_look(3N)
t_open: establish a transport endpoint ..... t_open(3N)
t_optmgmt: manage options for a transport endpoint ..... t_optmgmt(3N)
t_unbind: disable a transport endpoint ..... t_unbind(3N)
STREAMS module timod: Transport Interface cooperating ..... timod(7)
interface STREAMS module tirdwr: Transport Interface read/write ..... tirdwr(7)
t_sync: synchronize transport library ..... t_sync(3N)
the scheduler for the uucp file transport program ..... uusched(1M)
uucico: file transport program for the uucp system ..... uucico(1M)
establish a connection with another transport user ..... t_connect(3N)
data sent over a connection t_rcv: receive data or expedited ..... t_rcv(3N)
confirmation from a connect request t_rcvconnect: receive the ..... t_rcvconnect(3N)
disconnect t_rcvdis: retrieve information from ..... t_rcvdis(3N)
orderly release indication t_rcvrel: acknowledge receipt of an ..... t_rcvrel(3N)
t_rcvudata: receive a data unit ..... t_rcvudata(3N)
error indication t_rcvuderr: receive a unit data ..... t_rcvuderr(3N)
ftw: walk a file tree ..... ftw(3C)
tdelete, twalk: manage binary search trees ..... tsearch(3C)
atan, atan2: trigonometric/ trig: sin, cos, tan, asin, acos, ... trig(3M)
cos, tan, asin, acos, atan, atan2: trigonometric functions ..... trig(3M)
true, false: provide truth values ..... true(1)
truncate, ftruncate: truncate a file to a specified length ..... truncate(2SEQ)
to a specified length truncate, ftruncate: truncate a file ..... truncate(2SEQ)
u3b2, u3b5, vax: get processor type truth value ..... machid(1)
true, false: provide truth values ..... true(1)
ts: SCSI tape controller ..... ts(7)
manage binary search trees tsearch, tfind, tdelete, twalk: ..... tsearch(3C)
over a connection t_snd: send data or expedited data ..... t_snd(3N)
disconnect request t_snddis: send user-initiated ..... t_snddis(3N)
release t_sndrel: initiate an orderly ..... t_sndrel(3N)
t_sndudata: send a data unit ..... t_sndudata(3N)
tsort: topological sort ..... tsort(1)
library t_sync: synchronize transport ..... t_sync(3N)
interface tty: controlling terminal (gentty) ..... tty(7)
streamtty: Generic STREAMS tty driver interface ..... streamtty(7)
tty: get the name of the terminal ..... tty(1)
ld0: STREAMS tty line discipline ..... ld0(7)
between major/minor numbers and tty names ..... dev_to_ttyname(3C)
associated with a terminal or the/ ttydevname: get the device ..... ttydevname(1)
terminal ttyname, isatty: find name of a ..... ttyname(3C)
major/minor/ dev_to_ttyname, ttyname_to_dev: convert between ..... dev_to_ttyname(3C)
file of the current user ttyslot: find the slot in the utmp ..... ttyslot(3C)
endpoint t_unbind: disable a transport ..... t_unbind(3N)
parameters vmtune: tune system-wide virtual-memory ..... vmtune(1M)
tunefs: tune up an existing filesystem ..... tunefs(1M)
filesystem tunefs: tune up an existing ..... tunefs(1M)
examine and change virtual memory tuning parameters ..... vm_ctl(2SEQ)
/prdaily, prtacct, shutacct, startup, turnacct: shell procedures for/ ..... acctsh(1M)
tsearch, tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)

types: primitive system data types.....types(5)
types: primitive system data types.....types(5)
lptypes: table of printer types supported by the menu system.....lptypes(4)
/asctime, cftime, ascftime, mktime, tzset: convert date and time to/.....ctime(3C)
machid: i386, ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor type/.....machid(1)
machid: i386, ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor type/.....machid(1)
/i386, ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor type truth/.....machid(1)
uadmin: administrative control .....uadmin(1M)
uadmin: administrative control.....uadmin(2)
/endspent, fgetspent, lckpwdf, ulckpwdf: get shadow password file entry.....getspent(3X)
ulimit: get and set user limits .....ulimit(2)
mask umask: set and get file creation.....umask(2)
umask: set file-creation mode mask.....umask(1)
mount, umount: mount and unmount filesystems.....mount(1M)
umount: unmount a filesystem .....umount(2)
filesystems mountall, umountall: mount, unmount multiple.....mountall(1M)
system uname: get name of current operating.....uname(2)
uname: print name of current system .....uname(1)
expand data compress, uncompress, zcat: compress and .....compress(1)
unget: undo a previous get of an SCCS file .....unget(1)
SCCS file unget: undo a previous get of an .....unget(1)
input stream ungetc: push character back into .....ungetc(3S)
file uniq: report repeated lines in a.....uniq(1)
mktemp: make a unique filename .....mktemp(3C)
constants unistd: file header for symbolic.....unistd(4)
t_rcvudata: receive a data unit .....t_rcvudata(3N)
t_sndudata: send a data unit.....t_sndudata(3N)
t_rcvuderr: receive a unit data error indication.....t_rcvuderr(3N)
units: conversion program .....units(1)
mtio: UNIX magnetic tape interface .....mtio(7)
cu: call another UNIX system .....cu(1C)
link, unlink: link and unlink files and directories.....link(1M)
link, unlink: link and unlink files and directories.....link(1M)
unlink: remove directory entry.....unlink(2)
m_lock, m_unlock: lock, unlock locks .....m_lock(3PPS)
s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
space munmap: unmap part of the processs address .....munmap(2SEQ)
spellhist: unmatched spelling words file.....spellhist(4)
umount: unmount a filesystem.....umount(2)
mount, umount: mount and unmount filesystems.....mount(1M)
mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
pack, pcat, unpack: compress and expand files.....pack(1)
set, unset: set and unset environment variables in core or in files.....set(1F)
variables in core or in files set, unset: set and unset environment.....set(1F)
lsearch, lfind: linear search and update.....lsearch(3C)
touch: update access and modification times of a file.....touch(1)
make: maintain, update, and regenerate groups of programs .....make(1)
bdflush: update daemon.....bdflush(1M)
bdflush: update daemon.....bdflush(2SEQ)
sync: update superblock.....sync(2)
sync: update the superblock.....sync(1M)
from/ pwconv: installs and updates /etc/shadow with information.....pwconv(1M)

isismgr: ISIS file manager, updates text files in ISIS file format ..... isismgr(1M)
been up uptime: show how long system has..... uptime(1)
du: summarize disk usage ..... du(1M)
quota: display disk usage and limits..... quota(1)
retrieve a command description and usage examples..... usage(1)
description and usage examples usage: retrieve a command..... usage(1)
usclk: 32-bit microsecond counter..... usclk(7)
getusclk, usclk_init: get/init microsecond clock ..... getusclk(3SEQ)
get character login name of the user..... cuserid(3S)
logname: return login name of user..... logname(3X)
setup: initialize system for first user..... setup(1)
su: become superuser or another user..... su(1M)
a connection with another transport user..... t_connect(3N)
slot in the utmp file of the current user..... ttyslot(3C)
write: write to another user ..... write(1)
cpset: copy file and set mode, user, and group IDs..... cpset(1)
setuid, setgid: set user and group IDs..... setuid(2)
id: print user and group IDs and names..... id(1M)
crontab: user crontab file..... crontab(1)
addusr.def: add user defaults..... addusr_def(4)
/geteuid, getgid, getegid: get real user, effective user, real group/..... getuid(2)
environ: user environment ..... environ(5)
run commands performed for multi- user environment..... rc2(1M)
sulog: log of substitute user events ..... sulog(4)
generate disk accounting data by user ID..... diskusg(1M)
ulimit: get and set user limits..... ulimit(2)
tfmenush: assign menu system as user login shell..... tfmenush(1M)
menumsg: route user message to menu message queue..... menumsg(1M)
edquota: edit user quotas..... edquota(1M)
/getegid: get real user, effective user, real group, and effective/..... getuid(2)
t_snddis: send user-initiated disconnect request..... t_snddis(3N)
editor (variant of ex for casual users)..... edit(1)
the operating system for beginning users..... starter(1)
wall: write to all users ..... wall(1)
ut_delete_user: add/delete users in utmp file..... utadduser(3X)
mail, rmail: send mail to users or read mail..... mail(1)
login attempts /usr/adm/loginlog: log of failed/..... loginlog(4)
ustat: get filesystem statistics..... ustat(2)
add/delete users in utmp file ut_add_user, ut_add_user: ..... utadduser(3X)
utmp file ut_delete_user: add/delete users in ..... utadduser(3X)
introduction to system standalone utilities..... intro(8)
get_process_stats: get resource utilization information..... get_process_stats(2SEQ)
modification times utime: set file access and..... utime(2)
utmp, wtmp: utmp and wtmp entry formats..... utmp(4)
ut_delete_user: add/delete users in utmp file..... utadduser(3X)
endutent, utmpname, getutpid: access utmp file entry..... getut(3C)
ttyslot: find the slot in the utmp file of the current user..... ttyslot(3C)
formats utmp, wtmp: utmp and wtmp entry ..... utmp(4)
/pututline, setutent, endutent, utmpname, getutpid: access utmp file/..... getut(3C)
and permissions file uucheck: check the uucp directories..... uucheck(1M)
the uucp system uucico: file transport program for..... uucico(1M)
clean-up uucleanup: uucp spool directory ..... uucleanup(1M)

uucheck: check the uucp directories and permissions file.....uucheck(1M)
uusched: the scheduler for the uucp file transport program.....uusched(1M)
uucp: networking support files.....uucp(4)
uucleanup: uucp spool directory clean-up.....uucleanup(1M)
uustat: uucp status inquiry and job control.....uustat(1C)
file transport program for the uucp system.....uucico(1M)
UNIX-to-UNIX system/ uucp, uulog, uuname:.....uucp(1C)
speed, and line discipline uugetty: set terminal type, modes,.....uugetty(1M)
uucp, uulog, uuname: UNIX-to-UNIX/.....uucp(1C)
uucp, uulog, uuname: UNIX-to-UNIX system copy.....uucp(1C)
uuto, uupick: public UNIX-to-UNIX/.....uuto(1C)
file transport program uusched: the scheduler for the uucp.....uusched(1M)
control uustat: uucp status inquiry and job.....uustat(1C)
UNIX-to-UNIX system/ uuto, uupick: public.....uuto(1C)
with debugging on Uutry: try to contact remote system.....uutry(1M)
system command execution uux: UNIX-to-UNIX.....uux(1C)
requests uuxqt: execute remote command.....uuxqt(1M)
val: validate SCCS file.....val(1)
val: validate SCCS file.....val(1)
valloc: aligned memory allocator.....valloc(3SEQ)
abs: return integer absolute value.....abs(3C)
u3b5, vax: get processor type truth value.....machid(1)
getenv: return value for environment name.....getenv(3C)
floor, ceiling, remainder, absolute value functions.....floor(3M)
readlink: read value of a symbolic link.....readlink(2SEQ)
putenv: change or add value to environment.....putenv(3C)
current form field to its default values.....reset(1F)
true, false: provide truth values.....true(1)
values: machine-dependent values.....values(5)
values: machine-dependent values.....values(5)
print formatted output of a varargs argument list.....vprintf(3S)
list varargs: handle variable argument.....varargs(5)
varargs: handle variable argument list.....varargs(5)
fpathconf: get configurable pathname variables.....pathconf(2)
sysconf: get configurable system variables.....sysconf(2)
.environ, .pref, .variables: FACE files.....env(4)
unset: set and unset environment variables in core or in files.....set(1F)
edit: text editor (variant of ex for casual users).....edit(1)
proc_ctl: control over various process attributes.....proc_ctl(2SEQ)
/ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor type truth value.....machid(1)
vc: version control.....vc(1)
get option letter from argument vector.....getopt(3C)
readv: read vector from file.....readv(2SEQ)
writev: write vector on a file.....writev(2SEQ)
assert: verify program assertion.....assert(3X)
vc: version control.....vc(1)
get: get a version of an SCCS file.....get(1)
sccsdiff: compare two versions of an SCCS file.....sccsdiff(1)
virtual memory efficient way vfork: spawn new process in a.....vfork(2)
output of a varargs/ vprintf, vfprintf, vsprintf: print formatted.....vprintf(3S)
editor based on ex vi: screen-oriented (visual) display.....vi(1)
more: view a file one full screen at a time.....more(1)

map physical addresses into processs — virtual address space ..... pmap(7)
devtab: — virtual device table ..... devtab(4)
devdestroy: build and destroy — virtual devices ..... devbuild(1M)
vfork: spawn new process in a — virtual memory efficient way ..... vfork(2)
vm_ctl: examine and change — virtual memory tuning parameters ..... vm_ctl(2SEQ)
echo: put string on — virtual output ..... echo(1F)
controlling terminal fvhangup: — virtually hangup the specified ..... fvhangup(2SEQ)
vmtune: tune system-wide — virtual-memory parameters ..... vmtune(1M)
pmap: display or alter parameters of — virtual-to-physical mapping devices ..... pmap(1M)
vi: screen-oriented — visual display editor based on ex ..... vi(1)
memory tuning parameters — vm_ctl: examine and change virtual ..... vm_ctl(2SEQ)
virtual-memory parameters — vmtune: tune system-wide ..... vmtune(1M)
filesystem — volcopy: make literal copy of ..... volcopy(1M)
fs, inode: format of file system — volume ..... fs(4)
formatted output of a varargs/ — vprintf, vfprintf, vsprintf: print ..... vprintf(3S)
the controlling FMLI object — vsig: synchronize a co-process with ..... vsig(1F)
a varargs/ vprintf, vfprintf, — vsprintf: print formatted output of ..... vprintf(3S)
prtvtoc: print the — VTOC of a block device ..... prtvtoc(1M)
prtvtoc: print the — VTOC of a standalone disk device ..... prtvtoc(8)
mkvtoc: populate — VTOC on disks ..... mkvtoc(1M)
vtoc: — VTOC partitioning driver ..... vtoc(7)
vtoc: — vtoc: VTOC partitioning driver ..... vtoc(7)
s_wait_barrier: initialize barrier, — wait at barrier ..... s_wait_barrier(3PPS)
wait: await completion of process ..... wait(1)
wait, waitpid: — wait for a child process to stop or terminate ..... wait(2)
sigsuspend: — wait for a signal ..... sigsuspend(2)
process to stop or terminate — wait, waitpid: wait for a child ..... wait(2)
stop or terminate wait, — waitpid: wait for a child process to ..... wait(2)
ftw: — walk a file tree ..... ftw(3C)
wall: write to all users ..... wall(1)
wc: word count ..... wc(1)
wd: SSM SCSI disk controller ..... wd(7)
what: identify SCCS files ..... what(1)
whatis: describe what a command is ..... whatis(1)
whodo: — who is doing what ..... whodo(1M)
who: — who is on the system ..... who(1)
who: who is on the system ..... who(1)
whodo: who is doing what ..... whodo(1M)
vmtune: tune system- — wide virtual-memory parameters ..... vmtune(1M)
addformopt: add a form/SELECT — window option to a menu ..... addformopt(1M)
wc: — word count ..... wc(1)
fgetc, getw: get character or — word from a stream ..... getc(3S)
fputc, putw: put character or — word on a stream ..... putc(3S)
spellhist: unmatched spelling — words file ..... spellhist(4)
cd: change — working directory ..... cd(1)
chdir: change — working directory ..... chdir(2)
getcwd: get pathname of current — working directory ..... getcwd(3C)
pwd: — working directory name ..... pwd(1)
specific alarms and/or the — working indicator ..... indicator(1F)
interface — wp: SSM parallel line printer ..... wp(7)
lseek: move read/ — write file pointer ..... lseek(2)
tirdwr: Transport Interface read/ — write interface STREAMS module ..... tirdwr(7)

write: write on a file ..... write(2)
putpwent: write password file entry ..... putpwent(3C)
putspent: write shadow password file entry ..... putspent(3X)
wall: write to all users ..... wall(1)
write: write to another user ..... write(1)
writev: write vector on a file ..... writev(2SEQ)
write: write on a file ..... write(2)
write: write to another user ..... write(1)
writev: write vector on a file ..... writev(2SEQ)
open: open for reading or writing ..... open(2)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
accounting records fwtmp, wtmpfix: manipulate connect ..... fwtmp(1M)
and execute command xargs: construct argument list(s) ..... xargs(1)
multiplexor driver xs: Xylogics 781 terminal ..... xs(7)
tape interface xt: Xylogics 472 MULITBUS magnetic ..... xt(7)
xt: Xylogics 472 MULITBUS magnetic tape interface ..... xt(7)
xs: Xylogics 781 terminal multiplexor driver ..... xs(7)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
yacc: yet another compiler-compiler ..... yacc(1)
compress, uncompress, zcat: compress and expand data ..... compress(1)
zd: dual-channel disk controller ..... zd(7)
for dual-channel disk controller zdformat: standalone disk formatter ..... zdformat(8)
getzdinfo, getzdinfobyname: get zdinfo structure ..... getzdinfo(3SEQ)

NAME

intro – introduction to functions and libraries

DESCRIPTION

This section describes functions found in various libraries, other than those functions that directly invoke operating system primitives, which are described in Section 2. Major collections are identified by a letter after the section number:

(3C) These functions, together with those of Section 2 and those marked (3S), constitute the Standard C Library *libc*, which is automatically loaded by the C compiler, **cc**(1). The link editor **ld**(1) searches this library under the **–lc** option. Declarations for some of these functions may be obtained from *#include* files indicated on the appropriate pages.

(3M) These functions constitute the Math Library, *libm*. They are automatically loaded as needed by the FORTRAN compiler. They are not automatically loaded by the C compiler, **cc**(1); however, the link editor searches this library under the **–lm** option. Declarations for these functions may be obtained from the **#include** file *<math.h>*. Several generally useful mathematical constants are also defined there (see **math**(5)).

(3N) This contains sets of functions constituting the Network Services library. These sets provide protocol independent interfaces to networking services based on the service definitions of the OSI (Open Systems Interconnection) reference model. Application developers access the function sets that provide services at a particular level.

The function set contained in the library is:

TRANSPORT INTERFACE (TI) - provide the services of the OSI Transport Layer. These services provide reliable end-to-end data transmission using the services of an underlying network. Applications written using the TI functions are independent of the underlying protocols. Declarations for these functions may be obtained from the **#include** file **<tiuser.h>**. The link editor **ld**(1) searches this library under the **–lnsl_s** option.

(3PPS) The Sequent Parallel Programming Library. These routines take advantage of the parallel-processing capabilities of Sequent computers. Refer to **intro**(3pps) for more information about the Parallel Programming Library routines.

(3S) These functions constitute the standard I/O package (see **stdio**(3S)). These functions are in the library *libc*, already mentioned. Declarations for these functions may be obtained from the **#include** file *<stdio.h>*.

(3SEQ) These are Sequent-specific I/O functions in *libc*. Declarations for these

functions may be obtained from the **#include** file *<stdio.h>*.

(3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

DEFINITIONS

character Any bit pattern able to fit into a byte on the machine.

null character A character with value 0, represented in the C language as '\0'.

character array A sequence of characters.

null-terminated character array
A sequence of characters, the last of which is the null character.

string A designation for a null-terminated character array.

null string A character array containing only the null character.

NULL pointer The value that is obtained by casting **0** into a pointer. The C language guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return it to indicate an error.

NULL Defined as **0** in **<stdio.h>**; the user can include an appropriate definition if not using **<stdio.h>**.

generic function Many groups of FORTRAN intrinsic functions have generic function names that do not require explicit or implicit type declaration. The type of the function is determined by the type of its argument(s). For example, the generic function **max** returns an integer value if given integer arguments (**max0**), a real value if given real arguments (**amax1**), or a double-precision value if given double-precision arguments (**dmax1**).

netbuf In the Network Services library, **netbuf** is a structure used in various Transport Interface (TI) functions to send and receive data and information. It contains the following members:

```
unsigned int maxlen;
unsigned int len;
char  *buf;
```

buf Points to a user input and/or output buffer.

len Specifies the number of bytes contained in the buffer. If the structure is used for both input and output, the function replaces the user value of **len** on return.

maxlen Significant only when **buf** is used to receive output from the TI function. In this case, it specifies the physical size of the buffer, the maximum value of **len** that can be set by the function. If **maxlen** is not large enough to hold the returned information, an TBUFOVFLW error generally results. However, certain functions may

return part of the data and not generate an error.

FILES

/usr/lib
/usr/lib/libc.a
/usr/lib/libc_s.a
/usr/lib/libm.a
/usr/lib/lib77.a
/shlib/libc_s
/shlib/libnsl_s (3N)
/usr/lib/libnsl_s.a (3N)

SEE ALSO

ar(1), cc(1), ld(1), lint(1), nm(1), intro(2), intro(3PPS), stdio(3S), math(5)

DIAGNOSTICS

Functions in the C and Math Libraries (3C and 3M) may return the conventional values **0** or **±HUGE** (the largest-magnitude single-precision floating-point numbers; **HUGE** is defined in the *<math.h>* header file) when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable **errno** (see **intro**(2)) is set to the value **EDOM** or **ERANGE**.

WARNINGS

Many of the functions in the libraries call and/or refer to other functions and external variables described in this section and in Section 2, *System Calls*. If a program inadvertently defines a function or external variable with the same name, the presumed library version of the function or external variable may not be loaded. The **lint**(1) program checker reports name conflicts of this kind as "multiple declarations" of the names in question. Definitions for Sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the **-l** option. (For example, **-l***m* includes definitions for Section 3M, *Mathematical Library*.) Use of **lint** is highly recommended.

NAME

a64l, l64a – convert between long integer and base-64 ASCII string

SYNOPSIS

long a64l (s)
char *s;

char *l64a (l)
long l;

DESCRIPTION

These functions are used to maintain numbers stored in base-64 ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a digit in a radix-64 notation.

The characters used to represent digits are **.** for 0, **/** for 1, **0** through **9** for 2–11, **A** through **Z** for 12–37, and **a** through **z** for 38–63.

a64l takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. If the string pointed to by **s** contains more than six characters, **a64l** uses the first six.

a64l scans the character string from left to right, decoding each character as a 6 bit Radix 64 number.

l64a takes a **long** argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, **l64a** returns a pointer to a null string.

WARNINGS

The value returned by **l64a** is a pointer into a static buffer, the contents of which are overwritten by each call.

NAME

abort – generate an IOT fault

SYNOPSIS

int abort ( )

DESCRIPTION

abort does the work of **exit**(2), but instead of just exiting, **abort** causes SIGABRT to be sent to the calling process. If SIGABRT is neither caught nor ignored, all **stdio**(3S) streams are flushed prior to the signal being sent, and a core dump results.

abort returns the value of the **kill**(2) system call.

SEE ALSO

sdb(1), exit(2), kill(2), signal(2)

DIAGNOSTICS

If SIGABRT is neither caught nor ignored, and the current directory is writable, a core dump is produced and the message `abort - core dumped` is written by the shell.

NAME

abs – return integer absolute value

SYNOPSIS

int abs (i)
int i;

DESCRIPTION

abs returns the absolute value of its integer operand.

SEE ALSO

floor(3M)

WARNINGS

In two's-complement representation, the absolute value of the negative integer with largest magnitude is undefined. This error is not trapped and is ignored.

NAME

bsearch – binary search a sorted table

SYNOPSIS

#include <search.h>

char *bsearch (key, base, nel, sizeof (*key), compar)
char *key;
char *base;
unsigned nel;
int (*compar)( );

DESCRIPTION

bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. **key** points to a datum instance to be sought in the table. **base** points to the element at the base of the table. **nel** is the number of elements in the table. **compar** is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero as accordingly the first argument is to be considered less than, equal to, or greater than the second.

EXAMPLES

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```
#include <stdio.h>
#include <search.h>

#define     TABSIZE      1000

struct node {     /* these are stored in the table */
   char *string;
   int length;
};
struct node table[TABSIZE]; /* table to be searched */
   .
   .
   .
{
   struct node *node_ptr, node;
   int node_compare( ); /* routine to compare 2 nodes */
   char str_space[20];   /* space to read string into */
   .
   .
   .
   node.string = str_space;
   while (scanf("%s", node.string) != EOF) {
```

```
        node_ptr = (struct node *)bsearch((char *)(&node),
                (char *)table, TABSIZE,
                sizeof(struct node), node_compare);
        if (node_ptr != NULL) {
            (void)printf("string = %20s, length = %d\n",
                node_ptr->string, node_ptr->length);
        } else {
            (void)printf("not found: %s\n", node.string);
        }
    }
}
/*
    This routine compares two nodes based on an
    alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
char *node1, *node2;
{
    return (strcmp(
            ((struct node *)node1)->string,
            ((struct node *)node2)->string));
}
```

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although **bsearch** is declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

SEE ALSO

hsearch(3C), lsearch(3C), qsort(3C), tsearch(3C)

DIAGNOSTICS

A NULL pointer is returned if the key cannot be found in the table.

NAME

clock – report CPU time used

SYNOPSIS

long clock ( )

DESCRIPTION

clock returns the amount of CPU time (in microseconds) used since the first call to **clock**. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed **wait**(2), **pclose**(3S), or **system**(3S).

The resolution of the clock is 10 milliseconds.

SEE ALSO

times(2), wait(2), popen(3S), system(3S)

BUGS

The value returned by **clock** is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned wraps around after accumulating only 2147 seconds of CPU time (about 36 minutes).

NAME

conv: toupper, tolower, _toupper, _tolower, toascii – translate characters

SYNOPSIS

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

DESCRIPTION

toupper and **tolower** have as domain the range of **getc**(3S)– the integers from –1 through 255. If the argument of **toupper** represents a lowercase letter, the result is the corresponding uppercase letter. If the argument of **tolower** represents an uppercase letter, the result is the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

The macros **_toupper** and **_tolower**, are macros that accomplish the same thing as **toupper** and **tolower** but have restricted domains and are faster. **_toupper** requires a lowercase letter as its argument; its result is the corresponding uppercase letter. The macro **_tolower** requires an uppercase letter as its argument; its result is the corresponding lowercase letter. Arguments outside the domain cause undefined results.

toascii yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO

ctype(3C), getc(3S)

NAME

crypt, setkey, encrypt – generate hashing encryption

SYNOPSIS

```
char *crypt (key, salt)
char *key, *salt;

void setkey (key)
char *key;

void encrypt (block, ignored)
char *block;
int ignored;
```

DESCRIPTION

crypt is the password encryption function. It is based on a one way hashing encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.

key is a user's typed password. **salt** is a two-character string chosen from the set [a-zA-Z0-9.]; this string is used to perturb the hashing algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password. The first two characters are the salt itself.

The **setkey** and **encrypt** entries provide (rather primitive) access to the actual hashing algorithm. The argument of **setkey** is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is set into the machine. This is the key that will be used with the hashing algorithm to encrypt the string **block** with the function **encrypt**.

The argument to the **encrypt** entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the hashing algorithm using the key set by **setkey**. **ignored** is unused by **encrypt** but it must be present.

SEE ALSO

login(1), passwd(1), getpass(3C), passwd(4)

WARNINGS

The return value points to static data that are overwritten by each call.

NAME

ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset – convert date and time to string

SYNOPSIS

```
#include <sys/types.h>
#include <time.h>

char *ctime (clock)
time_t *clock;

struct tm *localtime (clock)
time_t *clock;

struct tm *gmtime (clock)
time_t *clock;

char *asctime (tm)
struct tm *tm;

int cftime(buf, fmt, clock)
char *buf, *fmt;
time_t *clock;

int ascftime (buf, fmt, tm)
char *buf, *fmt;
struct tm *tm;

time_t mktime (tm)
struct tm *tm;

extern long timezone, altzone;

extern int daylight;

extern char *tzname[2];

void tzset ( )
```

DESCRIPTION

ctime, **localtime**, and **gmtime** accept arguments of type **time_t** (declared in **<sys/types.h>**), pointed to by **clock**, representing the time in seconds since 00:00:00 UTC (Universal Time Code, formerly Greenwich Mean Time), January 1, 1970. **ctime** returns a pointer to a 26-character string in the following form. All the fields have constant width:

```
Fri Sep 13 00:00:00 1986\n\0
```

localtime and **gmtime** return pointers to **tm** structures, described below. **localtime** corrects for the main time zone and possible alternate ("Daylight Savings") time zone; **gmtime** converts directly to UTC, which is the time the operating system uses.

asctime converts a **tm** structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

Declarations of all the functions and externals, and the **tm** structure, are in the **<time.h>** header file. The structure declaration is:

```
structtm {
  int tm_sec;      /* seconds after the minute - [0, 59]*/
  int tm_min;      /* minutes after the hour - [0, 59]*/
```

```
        int tm_hour;    /* hour since midnight - [0, 23]*/
        int tm_mday;    /* day of the month - [1, 31]*/
        int tm_mon;     /* months since January - [0, 11]*/
        int tm_year;    /* years since 1900*/
        int tm_wday;    /* days since Sunday - [0, 6]*/
        int tm_yday;    /* days since January 1 - [0, 365]*/
        int tm_isdst;   /* flag for daylight savings time*/
    };
```

tm_isdst is non-zero if the alternate time zone is in effect.

cftime and **ascftime** provide the capabilities of **ctime** and **asctime**, respectively, as well as additional ones. **cftime** takes an integer of type **time_t** pointed to by **clock** and converts it to a character string. **ascftime** takes a pointer to a **tm** structure and converts it to a character string. In both functions, the characters are placed into the array pointed to by **buf** (plus a terminating **NULL** character) and the value returned is the number of such characters (not counting the terminating **NULL** character). **fmt** controls the format of the resulting string.

fmt is a character string that consists of field descriptors and text characters, like the C language **printf** statement. Each field descriptor consists of a % character followed by another character which specifies the replacement for the field descriptor. All other characters are copied from **fmt** into the result. The following field descriptors are supported:

%%	same as %
%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%d	day of month (01 - 31)
%D	date as %m/%d/%y
%e	day of month (1-31; single digits are preceded by a blank)
%h	abbreviated month name
%H	hour (00 - 23)
%I	hour (00 - 12)
%j	day number of year (001 - 366)
%m	month number (01 - 12)
%M	minute (00 - 59)
%n	same as \n
%p	ante meridian or post meridian
%r	time as %I:%M:%S %p
%R	time as %H:%M
%S	seconds (00 - 59)
%t	insert a tab
%T	time as %H:%M:%S
%U	week number of year (01 - 52), Sunday is the first day of week
%w	weekday number (Sunday = 0)
%W	week number of year (01 - 52), Monday is the first day of week
%x	Local specific date format
%X	Local specific time format

%y	year within century (00 - 99)
%Y	year as ccyy (e.g. 1986)
%Z	time zone name

The difference between %U and %W lies in which day is counted as the first of the week. Week number 01 is the first week with four or more January days in it.

The example below shows what the values in the **tm** structure would look like for Thursday, August 28, 1986 at 12:44:36 in New Jersey.

```
ascftime (buf, "%A %m %d %j", tm)
```

This example would result in the buffer containing `"Thursday Aug 28 240"`.

If **fmt** is **(char*)0**, the value of the environment variable **CFTIME** is used. If **CFTIME** is undefined or empty, a default format is used. The default format string is taken from the file that contains the date and time strings associated with the then current language [see below for details on changing the current language and **cftime**(4) for a description of the structure of these files].

The user can request that the output of **cftime** and **ascftime** be in a specific language by setting the environment variable **LANGUAGE** to the desired language. If **LANGUAGE** is empty, unset or set to an unsupported language, the last language requested is used (the default is the **usa-english** strings).

mktime converts the broken-down time (expressed as local time in the structure pointed to by *tm), into a calendar time value with the same encoding as that of the values returned by the **time** function. The original values of the **tm_wday** and **tm_yday** components of the structure are ignored, and the original values of the other components are not restricted to the ranges indicated above. Thus, a positive value for **tm_isdat** causes **mktime** to presume that Daylight Savings Time is in effect, and a zero value for **tm_isdat** causes **mktime** to presume that Daylight Savings Time is not in effect. A negative value for **tm_isdat** causes **mktime** to attempt to determine if Daylight Savings Time is in effect for the specified time. On successful completion, the values of the **tm_wday** and **tm_yday** components of the structure are set appropriately, and the other components are set to represent the calendar time, but with their values forced to the ranges indicated above. **mktime** returns the converted calendar time in seconds since 00:00:00 UTC, January 1, 1970, encoded as a value of type **time_t**. If the calendar time cannot be represented, **mktime** returns the value -1 (**time_t**). The following **mktime** example answers the question, "What day of the week is July 4, 2001?":

```
#include <stdio.h>
#include <time.h>
static cont char *const wday[] = {
      "Sunday", "Monday", "Tuesday", "Wednesday",
      "Thursday", "Friday", "Saturday", "Sunday", "-unknown-"
};
struct tm time_str;

time_str.tm_year  = 2001-1900;
time_str.tm_mon   =7 - 1;
time_str.tm_mday  = 4;
```

```
time_str.tm_hour  = 0;
time_str.tm_min   = 0;
time_str.tm_sec   = 1;
time_str.tm_isdst = -1;
if (mktime (&time_str)    == -1)
      time_str.tm_wday  = 7;
printf ("%s0, wday[time_str.tm_wday]);
```

The external **long** variable **timezone** contains the difference, in seconds, between UTC and the main time zone; the external **long** variable **altzone** contains the difference, in seconds, between UTC and the alternate time zone; both, **timezone** and **altzone** default to 0 (UTC). The external variable **daylight** is non-zero if an alternate time zone exists. The time zone names are contained in the external variable **tzname**, which by default is set to

```
char *tzname[2] = { "UTC", "    " };
```

The functions know about the peculiarities of this conversion for various time periods for the U.S.A (specifically, the years 1974, 1975, and 1987). The functions handle the new daylight savings time starting with the first Sunday in April, 1987.

tzset uses the contents of the environment variable **TZ** to override the value of the different external variables. The syntax of **TZ** can be described as follows:

TZ	→	*zone*
		| *zone signed_time*
		| *zone signed_time zone*
		| *zone signed_time zone dst*
zone	→	*letter letter letter*
signed_time	→	*sign time*
		| *time*
time	→	*hour*
		| *hour : minute*
		| *hour : minute : second*
dst	→	*signed_time*
		| *signed_time ; dst_date , dst_date*
		| *; dst_date , dst_date*
dst_date	→	*date / time*
date	→	**J***julianone*
		| *julianzero*
		| **M***month.week.day*
month	→	*1* | *2* | .. | *12*
week	→	*1* | *2* | *3* | *4* | *5*
day	→	*0* | *1* | *2* | *3* | *4* | *5* | *6*
letter	→	*a* | *A* | *b* | *B* | ... | *z* | *Z*
hour	→	*00* | *01* | ... | *23*
minute	→	*00* | *01* | ... | *59*
second	→	*00* | *01* | ... | *59*

julianzero → *0* | *1* | ... | *365*
julianone → *1* | *2* | ... | *366*
sign → – | +

tzset scans the contents of the environment variable and assigns the different fields to the respective variable. For example, the setting for New Jersey in 1986 could be

```
"EST5EDT4;117/2:00:00,299/2:00:00"
```

or simply

```
EST5EDT
```

A southern hemisphere setting such as the Cook Islands could be

```
"KDT9:30KST10:00;64/5:00,303/20:00"
```

When the longer format is used, the variable must be surrounded by double quotes as shown. For more details, see **timezone**(4) and **environ**(5). In the longer version of the New Jersey example of **TZ**, **tzname**[0] is `EST`, **timezone** is set to `5*60*60`, **tzname**[1] is `EDT`, **altzone** is set to `4*60*60`, the starting date of the alternate time zone is the 117th day at 2 AM, the ending date of the alternate time zone is the 299th day at 2 AM, and **daylight** is set to non-zero. Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year are used and the time is 2 AM. If the start and end dates are provided but the time is not provided, the time is midnight. The effects of **tzset** are thus to change the values of the external variables **timezone, altzone, daylight** and **tzname. tzset** is called by **localtime** and may also be called explicitly by the user.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, via the local */etc/profile file* [see **profile**(4)].

FILES

/lib/cftime – directory that contains the language specific printable files

SEE ALSO

time(2), strftime(3C), getenv(3C), putenv(3C), printf(3S), cftime(4), profile(4), timezone(4), environ(5)

WARNINGS

The return values for **ctime, localtime** and **gmtime** point to static data whose content is overwritten by each call.

Setting the time during the interval of change from **timezone** to **altzone** or vice versa can produce unpredictable results.

The system administrator must change the Julian start and end days annually if the full form of the **TZ** variable is specified.

NAME

isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper, toascci, _tolower, _toupper, setchrclass – character handling

SYNOPSIS

```
#include <ctype.h>

int isdigit (c);
int c;

. . .

tolower(c)
int c;

. . .

int setchrclass (chrclass)
char *chrclass;
```

DESCRIPTION

The character classification macros listed below return non-zero for true, zero for false. **isascii** is defined on all integer values; the rest are defined on valid members of the character set and on the single value **EOF** [see **stdio**(3S)] (guaranteed not to be a character set member).

isdigit	tests for the digits 0 through 9.
isxdigit	tests for any character for which **isdigit** is true or for the letters **a** through *f* or *A* through *F*.
islower	tests for any lowercase letter as defined by the character set.
isupper	tests for any uppercase letter as defined by the character set.
isalpha	tests for any character for which **islower** or **isupper** is true and possibly any others as defined by the character set.
isalnum	tests for any character for which **isalpha** or **isdigit** is true.
isspace	tests for a space, horizontal-tab, carriage return, newline, vertical-tab, or form-feed.
iscntrl	tests for "control characters" as defined by the character set.
ispunct	tests for any character other than the ones for which **isalnum**, **iscntrl**, or **isspace** is true or space.
isprint	tests for a space or any character for which **isalnum** or **ispunct** is true or other "printing character" as defined by the character set.
isgraph	tests for any character for which **isprint** is true, except for space.
isascii	tests for an ASCII character (a non-negative number less than 0200.)

The conversion functions and macros translate a character from lowercase (uppercase) to uppercase (lowercase).

tolower if the character is one for which **isupper** is true and there a corresponding lowercase character, **tolower** returns the corresponding lowercase character. Otherwise, the character is returned unchanged.

toupper if the character is one for which **islower** is true and there is a corresponding uppercase character, **toupper** returns the corresponding uppercase character. Otherwise, the character is returned unchanged.

toascii turns off the bits that are not part of the ASCII character set.

_tolower returns the lowercase representation of a character for which **isupper** is true, otherwise undefined.

_toupper returns the uppercase representation of a character for which **islower** is true, otherwise undefined.

toupper and **tolower** have as domain the range of **getc**(3S)– the integers from –1 through 255. All other arguments in the domain are returned unchanged. The conversion macros have the same functionality of the functions on valid input, but the macros are faster because they do not do range checking.

All the character classification macros and the conversion functions and macros do a table lookup.

setchrclass initializes the table used by these functions and macros to a specific character classification set. **setchrclass** uses the value of its argument or the value of the environment variable CHRCLASS as the name of the datafile containing the information for the desired character set. These datafiles are searched for in the special directory /lib/chrclass.

If **chrclass** is (**char***)0, the value of the environment variable CHRCLASS is used. If CHRCLASS is not set or is undefined, the table retains its current value, which at initialization time is **ASCII**.

FILES

/lib/chrclass – directory containing the datafiles for **setchrclass**

SEE ALSO

chrtbl(1), stdio(3S), ascii(5), environ(5)

DIAGNOSTICS

If the argument to any of the character handling macros is not in the domain of the function, the result is undefined.

If **setchrclass** does not successfully fill the table, the table will not change (initially ASCII) and –1 is returned. If successful, **setchrclass** returns 0.

NAME

dev_to_ttyname, ttyname_to_dev – convert between major/minor numbers and tty names

SYNOPSIS

```
char *dev_to_ttyname(dev)
dev_t dev;

int ttyname_to_dev(name, dev)
char *name;
dev_t *dev;
```

DESCRIPTION

These functions do conversions between device major and minor numbers and *tty* pathnames. The path is guaranteed to be unique for major numbers between 0 and 1023 inclusive (10 bits) and minor numbers between 0 and 16383 inclusive (14 bits), or 16,777,216 entries. There is a one-to-one mapping between devices and their names.

dev_to_ttyname returns a pointer to a NULL-terminated string of the form */dev/tty<subdir>/<basename>*, where *basename* is a unique four character string and *subdir* is two characters. The first and third characters of *basename* determine the directory name for *subdir*. For example, in */dev/ttyac/abcd*), the first character, *a*, and the third character, *c*, are appended to *tty* to determine the *subdir* directory name *ttyac*. Using the first and third symbols from *basename* causes minor numbers to rotate between 4 directories every 64 entries. No other major numbers use those 4 directories.

Not counting the string */dev/*, the resulting 10 character path names are suitable for use in **utmp** *ut_line* field. The unique 4 character *basename* can be used in the **utmp** *ut_id* field. The entire path can be constructed from the unique *basename*.

ttyname_to_dev takes any potiential path to a *tty* device, extracts the *basename* and calculates the associated device. The device is then returned in **dev**.

To arrive at the *subdir–basename* combination, the lower 10 bits of the major number and lower 14 bits of the minor number are converted into four 6 bit quantities. Each 6 bit value is converted into a unique 4 character name using base 64 arithmetic.

Base 64 digits are arbitrarily represented by the symbols [A-Za-z0-9_+] where 'A' == 0, 'a' == 26, '0' == 52, underscore (_) == 62, and plus sign (+) == 63. The number AAAA has the value 0 and AAA+ has the value 63.

DIAGNOSTICS

Upon successful completion, **ttyname_to_dev** returns 0. If the *basename* of the input string is greater than four non-NULL characters or the *basename* contains symbols not found in the regular expression [A-Za-z0-9_+], a value of -1 is returned.

There is no error condition for **dev_to_ttyname**. It silently truncates **dev** to 10 bits of major and 14 bits of minor.

EXAMPLES

The following table gives an example of the directory names used for certain major numbers:

major number	directory	minor ranges 0 to 255	256 to 511
0	ttyAA	AAAA to AAA+	ABAA to ABA+
	ttyAB	AABA to AAB+	ABBA to ABB+
	ttyAC	AACA to AAC+	ABCA to ABC+
	ttyAD	AADA to AAD+	ABDA to ABD+
6	ttyGA	GAAA to GAA+	GBAA to GBA+
	ttyGB	GABA to GAB+	GBBA to GBB+
	ttyGC	GACA to GAC+	GBCA to GBC+
	ttyGD	GADA to GAD+	GBDA to GBD+

The tty device names for the 16 lines of a controller with major number 6 are:

major=6, minor=0	/dev/ttyGA/GAAA
major=6, minor=1	/dev/ttyGA/GAAB
major=6, minor=2	/dev/ttyGA/GAAC
.	.
.	.
major=6, minor=15	/dev/ttyGA/GAAP

SEE ALSO

ttyname (3c), ttydevname (1)

BUGS

These routines are designed to allow speculation, thus major numbers are not checked for validity before conversion. For example, major=0 and major=1024 return the same path.

Only the *basename* of *name* is checked, so an obviously bad path may still return a valid **dev**. For example, the path */dev/hello/world/ABCD* would show *ABCD* as a valid basename.

NAME

dial – establish an out-going terminal line connection

SYNOPSIS

include <dial.h>

int dial (call)
CALL call;

void undial (fd)
int fd;

DESCRIPTION

dial returns a file-descriptor for a terminal line open for read/write. The argument to **dial** is a **CALL** structure (defined in the **<dial.h>** header file).

When finished with the terminal line, the calling program must invoke **undial** to release the semaphore that has been set during the allocation of the terminal device.

The definition of **CALL** in the **<dial.h>** header file is:

```
typedef struct {
  struct termio *attr;/* pointer to termio attribute struct */
  int   baud;     /* transmission data rate */
  int   speed;    /* 212A modem: low=300, high=1200 */
  char  *line;    /* device name for out-going line */
  char  *telno;   /* pointer to tel-no digits string */
  int   modem;    /* specify modem control for direct lines */
  char  *device;  /* unused */
  int   dev_len;  /* unused */
} CALL;
```

The **CALL** element **speed** is intended only for use with an outgoing dialed call. The value should match the modem speed setting. The typical values are: 110, 300, 600, 1200, 2400, 4800, 9600, and 19200. The **CALL** element **baud** is for the desired transmission baud rate. For example, one might set **baud** to 110 and **speed** to 300 (or 1200). However, if **speed** is set to 1200, **baud** must be set to high (1200).

If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the **line** element in the **CALL** structure. Legal values for such terminal device names are kept in the *Devices* file. In this case, the value of the **baud** element should be set to -1. This will cause **dial** to determine the correct value from the *Devices* file.

The **telno** element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of these characters:

0-9	dial 0-9
*	dial *
#	dial #
=	wait for secondary dail tone
–	delay for approximately 4 seconds

The **CALL** element **modem** is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The **CALL** element **attr** is a pointer to a **termio** structure, as defined in the **<termio.h>** header file. A NULL value for this pointer element may be passed to the **dial** function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This is often important for certain attributes such as parity and baud-rate.

The **CALL** elements device and **dev_len** are no longer used. They are retained in the **CALL** structure for compatibility reasons.

FILES

/usr/lib/uucp/Devices
/usr/lib/uucp/Systems
/usr/spool/uucp/LCK.*tty-device*

SEE ALSO

uucp(1C), alarm(2), read(2), write(2), termio(7)

DIAGNOSTICS

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the **<dial.h>** header file.

```
INTRPT     -1     /* interrupt occurred */
D_HUNG     -2     /* dialer hung (no return from write) */
NO_ANS     -3     /* no answer within 10 seconds */
ILL_BD     -4     /* illegal baud-rate */
A_PROB     -5     /* acu problem (open() failure) */
L_PROB     -6     /* line problem (open() failure) */
NO_Ldv     -7     /* can't open Devices file */
DV_NT_A    -8     /* requested device not available */
DV_NT_K    -9     /* requested device not known */
NO_BD_A    -10    /* no device available at requested baud */
NO_BD_K    -11    /* no device known at requested baud */
DV_NT_E    -12    /* requested speed does not match */
BAD_SYS    -13    /* system not in Systems file*/
```

WARNINGS

Including the **<dial.h>** header file automatically includes the **<termio.h>** header file.

The above routine uses **<stdio.h>**, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.

BUGS

An **alarm**(2) system call for 3600 seconds is made (and caught) within the **dial** module for the purpose of "touching" the *LCK..* file and constitutes the device allocation semaphore for the terminal device. Otherwise, **uucp**(1C) may simply delete the *1LCK..* entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a **read**(2) or **write**(2) system call, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from **reads** should be checked for **(errno==EINTR)**, and the **read** possibly reissued.

NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudorandom numbers

SYNOPSIS

```
double drand48 ( )

double erand48 (xsubi)
unsigned short xsubi[3];

long lrand48 ( )

long nrand48 (xsubi)
unsigned short xsubi[3];

long mrand48 ( )

long jrand48 (xsubi)
unsigned short xsubi[3];

void srand48 (seedval)
long seedval;

unsigned short *seed48 (seed16v)
unsigned short seed16v[3];

void lcong48 (param)
unsigned short param[7];
```

DESCRIPTION

This family of functions generates pseudorandom numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions **drand48** and **erand48** return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions **lrand48** and **nrand48** return non-negative long integers uniformly distributed over the interval $[0, 2^{31})$.

Functions **mrand48** and **jrand48** return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$.

Functions **srand48**, **seed48** and **lcong48** are initialization entry points, one of which should be invoked before either **drand48**, **lrand48** or **mrand48** is called. (Although it is not recommended practice, constant default initializer values are supplied automatically if **drand48**, **lrand48** or **mrand48** is called without a prior call to an initialization entry point.) Functions **erand48**, **nrand48** and **jrand48** do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula

$$X_{n+1}=(aX_n+c)_{\mathrm{mod}\,m}\; n \geq 0.$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless **lcong48** has been invoked, the multiplier value a and the addend value c are given by

$$a=5\mathrm{DEECE66D}_{16}=273673163155_8$$
$$c=\mathrm{B}_{16}=13_8.$$

The value returned by any of the functions **drand48**, **erand48**, **lrand48**, **nrand48**, **mrand48** or **jrand48** is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

The functions **drand48**, **lrand48** and **mrand48** store the last 48-bit X_i generated in an internal buffer, and must be initialized prior to being invoked. The functions **erand48**, **nrand48** and **jrand48** require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. These routines do not have to be initialized; the calling program must place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, functions **erand48**, **nrand48** and **jrand48** allow separate modules of a large program to generate several independent streams of pseudorandom numbers, i.e., the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function **srand48** sets the high-order 32 bits of X_i its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function **seed48** sets the value of X_i to the 48-bit value specified in the argument array. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by **seed48**, and a pointer to this buffer is the value returned by **seed48**. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last X_i value, and then use this value to reinitialize via **seed48** when the program is restarted.

The initialization function **lcong48** allows the user to specify the initial X_i, the multiplier value a, and the addend value c. Argument array elements **param[0-2]** specify X_i, **param[3-5]** specify the multiplier a, and **param[6]** specifies the 16-bit addend c. After **lcong48** has been called, a subsequent call to either **srand48** or **seed48** will restore the "standard" multiplier and addend values, a and c, specified on the previous page.

NOTES

The source code for the portable version can be used on computers which do not have floating-point arithmetic. In such a situation, functions **drand48** and **erand48** are replaced by the two new functions below.

```
long irand48 (m)
unsigned short m;

long krand48 (xsubi, m)
unsigned short xsubi[3], m;
```

Functions **irand48** and **krand48** return non-negative long integers uniformly distributed over the interval [0, m-1].

SEE ALSO

rand(3C)

NAME

ecvt, fcvt, gcvt – convert floating-point number to string

SYNOPSIS

```
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt (value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

DESCRIPTION

ecvt converts **value** to a NULL-terminated string of **ndigit** digits and returns a pointer thereto. The high-order digit is non-zero, unless the value is zero. The low-order digit is rounded. The position of the decimal point relative to the beginning of the string is stored indirectly through **decpt** (negative means to the left of the returned digits). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by **sign** is non-zero, otherwise it is zero.

fcvt is identical to **ecvt**, except that the correct digit has been rounded for **printf** *%f* (FORTRAN F-format) output of the number of digits specified by **ndigit**.

gcvt converts the **value** to a null-terminated string in the array pointed to by **buf** and returns **buf**. It attempts to produce **ndigit** significant digits in FORTRAN F-format if possible, otherwise E-format, ready for printing. A minus sign, if there is one, or a decimal point will be included as part of the returned string. Trailing zeros are suppressed.

SEE ALSO

printf(3S)

BUGS

The values returned by **ecvt** and **fcvt** point to a single static data array whose content is overwritten by each call. The static data array is limited to 80 characters in length, so any results greater than 80 characters may yield unpredictable results.

NAME

end, etext, edata – last locations in program

SYNOPSIS

```
extern end;
extern etext;
extern edata;
```

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of **etext** is the first address above the program text, **edata** above the initialized data region, and **end** above the uninitialized data region.

When execution begins, the program break (the first location beyond the data) coincides with **end**, but the program break may be reset by the routines of **brk**(2), **malloc**(3C), standard input/output (**stdio**(3S)), the profile (**-p**) option of **cc**(1), and so on. Thus, the current value of the program break should be determined by **sbrk (char *)(0)** (see **brk**(2)).

SEE ALSO

cc(1), brk(2), malloc(3C), stdio(3S)

NAME

fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky – IEEE floating-point environment control

SYNOPSIS

```
include <ieeefp.h>

typedef enum {
        FP_RN=0,        /* round to nearest */
        FP_RP,          /* round to plus */
        FP_RM,          /* round to minus */
        FP_RZ,          /* round to zero (truncate) */
        } fp_rnd;

        fp_rnd fpgetround();

        fp_rnd fpsetround(rnd_dir)
        fp_rnd rnd_dir;

        define  fp_except       int
        define  FP_X_INV        0x01    /* invalid operation exception*/
        define  FP_X_DNML       0x02    /* denormalization exception*/
        define  FP_X_DZ         0x04    /* divide-by-zero exception*/
        define  FP_X_OFL        0x08    /* overflow exception*/
        define  FP_X_UFL        0x10    /* underflow exception*/
        define  FP_X_IMP        0x20    /* imprecise (loss of precision)*/

        fp_except fpgetmask();

        fp_except fpsetmask(mask);
        fp_except mask;

        fp_except fpgetsticky();

        fp_except fpsetsticky(sticky);
        fp_except sticky;
```

DESCRIPTION

There are six floating-point exceptions: divide-by-zero, overflow, underflow, imprecise (inexact) result, denormalization, and invalid operation. When a floating-point exception occurs, the corresponding sticky bit is set (1), and if the mask bit is enabled (1), the trap takes place. These routines let the user change the behavior on occurrence of any of these exceptions, as well as change the rounding mode for floating-point operations.

fpgetround () returns the current rounding mode.

fpsetround () sets the rounding mode and returns the previous rounding mode.

fpgetmask () returns the current exception masks.

fpsetmask () sets the exception masks and returns the previous setting.

fpgetsticky () returns the current exception sticky flags.

fpsetsticky () sets (clears) the exception sticky flags and returns the previous setting.

The default environment is:

Rounding mode set to nearest(FP_RN),
Divide-by-zero,
Floating point overflow, and
Invalid operation traps enabled.

SEE ALSO

isnan(3C)

WARNINGS

fpsetsticky () modifies all sticky flags. **fpsetmask** () changes all mask bits.

Both C and FORTRAN require truncation (round to zero) for floating-point to integral conversions. The current rounding mode has no effect on these conversions.

One must clear the sticky bit to recover from the trap and to proceed. If the sticky bit is not cleared before the next trap occurs, a wrong exception type may be signaled.

For the same reason, when calling **fpsetmask()** the user should make sure that the sticky bit corresponding to the exception being enabled is cleared.

NAME

frexp, ldexp, modf – manipulate parts of floating-point numbers

SYNOPSIS

```
double frexp (value, eptr)
double value;
int *eptr;

double ldexp (value, exp)
double value;
int exp;

double modf (value, iptr)
double value, *iptr;
```

DESCRIPTION

Every non-zero number can be written uniquely as x *2 to the nth power where the *mantissa* (fraction) **x** is in the range $0.5 \leq | x | < 1.0$, and the *exponent* **n** is an integer. **frexp** returns the mantissa of a double **value**, and stores the exponent indirectly in the location pointed to by **eptr**. If **value** is zero, both results returned by **frexp** are zero.

ldexp returns the quantity **value** *2 to the **exp** power.

modf returns the signed fractional part of **value** and stores the integral part indirectly in the location pointed to by **iptr**.

DIAGNOSTICS

If **ldexp** would cause overflow, ±**HUGE** (defined in **<math.h>**) is returned (according to the sign of **value**), and **errno** is set to **ERANGE**.

If **ldexp** would cause underflow, zero is returned and **errno** is set to **ERANGE**.

NAME

ftw – walk a file tree

SYNOPSIS

```
#include <ftw.h>

int ftw (path, fn, depth)
char *path;
int (*fn) ( );
int depth;
```

DESCRIPTION

ftw recursively descends the directory hierarchy rooted in **path**. For each object in the hierarchy, **ftw** calls **fn**, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a **stat** structure (see **stat**(2)) containing information about the object, and an integer. Possible values of the integer, defined in the **<ftw.h>** header file, are **FTW_F** for a file, **FTW_D** for a directory, **FTW_DNR** for a directory that cannot be read, and **FTW_NS** for an object for which **stat** could not successfully be executed. If the integer is **FTW_DNR**, descendants of that directory will not be processed. If the integer is **FTW_NS**, the **stat** structure will contain garbage. An example of an object that would cause **FTW_NS** to be passed to **fn** would be a file in a directory with read but without execute (search) permission.

ftw visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of **fn** returns a non-zero value, or some error is detected within **ftw** (such as an I/O error). If the tree is exhausted, **ftw** returns zero. If **fn** returns a non-zero value, **ftw** stops its tree traversal and returns whatever value was returned by **fn**. If **ftw** detects an error, it returns –1, and sets the error type in **errno**.

ftw uses one file descriptor for each level in the tree. The **depth** argument limits the number of file descriptors so used. If **depth** is zero or negative, the effect is the same as if it were 1. **depth** must not be greater than the number of file descriptors currently available for use. **ftw** will run more quickly if **depth** is at least as large as the number of levels in the tree.

SEE ALSO

stat(2), malloc(3C)

BUGS

Because **ftw** is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

WARNINGS

ftw uses **malloc**(3C) to allocate dynamic storage during its operation. If **ftw** is forcibly terminated, such as by **longjmp** being executed by **fn** or an interrupt routine, **ftw** will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have **fn** return a non-zero value at its next invocation.

NAME

getcwd – get pathname of current working directory

SYNOPSIS

char *getcwd (buf, size)
char *buf;
int size;

DESCRIPTION

getcwd returns a pointer to the current directory pathname. The value of **size** must be at least two greater than the length of the pathname to be returned.

If **buf** is a NULL pointer, **getcwd** obtains **size** bytes of space using **malloc**(3C). In this case, the pointer returned by **getcwd** may be used as the argument in a subsequent call to **free.**

EXAMPLES

```
void exit(), perror();
.
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
      perror("getcwd");
      exit(2);
}
printf("%s\n", cwd);
```

SEE ALSO

malloc(3C)

DIAGNOSTICS

Returns NULL with **errno** set if **size** is not large enough, or if an error occurs in a lower-level function.

NAME

getenv – return value for environment name

SYNOPSIS

char *getenv (name)
char *name;

DESCRIPTION

getenv searches the environment list (see **environ**(5)) for a string of the form

name = value

and returns a pointer to the **value** in the current environment if such a string is present, otherwise a NULL pointer.

SEE ALSO

exec(2), putenv(3C), environ(5)

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent – get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent ( )

struct group *getgrgid (gid)
gid_t gid;

struct group *getgrnam (name)
char *name;

void setgrent ( )

void endgrent ( )

struct group *fgetgrent (f)
FILE *f;
```

DESCRIPTION

getgrent, getgrgid and **getgrnam** each return pointers to an object with the following structure containing the broken-out fields of a line in the */etc/group* file. Each line contains a "group" structure, defined in the **<grp.h>** header file.

```
struct group {
        char   *gr_name;  /* the name of the group */
        char   *gr_passwd;/* the encrypted group password */
        gid_t  gr_gid;    /* the numerical group ID */
        char   **gr_mem;  /* vector of pointers to member names *
};
```

getgrent when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. **getgrgid** searches from the beginning of the file until a numerical group id matching **gid** is found and returns a pointer to the particular structure in which it was found. **getgrnam** searches from the beginning of the file until a group name matching **name** is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **setgrent** has the effect of rewinding the group file to allow repeated searches. **endgrent** may be called to close the group file when processing is complete.

fgetgrent returns a pointer to the next group structure in the stream **f**, which matches the format of */etc/group*.

FILES

/etc/group

SEE ALSO

getlogin(3C), getpwent(3C), group(4)

DIAGNOSTICS

A NULL pointer is returned on EOF or error.

WARNINGS

The above routines use *<stdio.h>*, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getlogin – get login name

SYNOPSIS

char *getlogin ( );

DESCRIPTION

getlogin returns a pointer to the login name as found in */etc/utmp*. It may be used in conjunction with **getpwnam** to locate the correct password file entry when the same user ID is shared by several login names.

If **getlogin** is called within a process that is not attached to a terminal, it returns a **NULL** pointer. The correct procedure for determining the login name is to call **cuserid**, or to call **getlogin** and if it fails to call **getpwuid**.

FILES

/etc/utmp

SEE ALSO

cuserid(3S), getgrent(3C), getpwent(3C), utmp(4)

DIAGNOSTICS

Returns the **NULL** pointer if **name** is not found.

WARNINGS

The return values point to static data whose content is overwritten by each call.

NAME

getopt – get option letter from argument vector

SYNOPSIS

```
int getopt (argc, argv, optstring)
int argc;
char **argv, *opstring;

extern char *optarg;
extern int optind, opterr;
```

DESCRIPTION

getopt returns the next option letter in **argv** that matches a letter in **optstring**. It supports all the rules of the command syntax standard (see **intro**(1)). So all new commands will adhere to the command syntax standard, they should use **getopts**(1) or **getopt**(3C) to parse positional parameters and check for options that are legal for that command.

optstring must contain the option letters the command using **getopt** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

optarg is set to point to the start of the option-argument on return from **getopt**.

getopt places in **optind** the **argv** index of the next argument to be processed. **optind** is external and is initialized to **1** before the first call to **getopt**.

When all options have been processed (i.e., up to the first non-option argument), **getopt** returns **–1**. The special option "—" may be used to delimit the end of the options; when it is encountered, **–1** will be returned, and "—" will be skipped.

DIAGNOSTICS

getopt prints an error message on standard error and returns a question mark (**?**) when it encounters an option letter not included in **optstring** or no option-argument after an option that expects one. This error message may be disabled by setting **opterr** to **0**.

EXAMPLES

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the option **o**, which requires an option-argument:

```
main (argc, argv)
int argc;
char **argv;
{
        int c;
        extern char *optarg;
        extern int optind;

        .

        .
```

```
.
while ((c = getopt(argc, argv, "abo:")) != -1)
      switch (c) {
      case 'a':
            if (bflg)
                  errflg++;
            else
                  aflg++;
            break;
      case 'b':
            if (aflg)
                  errflg++;
            else
                  bproc( );
            break;
      case 'o':
            ofile = optarg;
            break;
      case '?':
            errflg++;
      }
if (errflg) {
      (void)fprintf(stderr, "usage: . . . ");
      exit (2);
}
for ( ; optind < argc; optind++) {
      if (access(argv[optind], 4)) {

.

.

.
}
```

WARNINGS

Although the following command syntax rule (see **intro**(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

```
cmd -aboxxx file    (Rule 5 violation: options with
    option-arguments must not be grouped with other options)
cmd -ab -oxxx file  (Rule 6 violation: there must be
    white space after an option that takes an option-argument)
```

SEE ALSO

getopts(1), intro(1)

Changing the value of the variable **optind**, or calling **getopt** with different values of **argv**, may lead to unexpected results.

NAME

getpass – read a password

SYNOPSIS

char *getpass (prompt)
char *prompt;

DESCRIPTION

getpass reads up to a newline or **EOF** from the file */dev/tty*, after prompting on the standard error output with the null-terminated string **prompt** and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters. If */dev/tty* cannot be opened, a **NULL** pointer is returned. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

FILES

/dev/tty

WARNINGS

The above routine uses **<stdio.h>**, which causes it to increase the size of programs not otherwise using standard I/O more than might be expected.

The return value points to static data whose content is overwritten by each call.

NAME

getpw – get name from UID

SYNOPSIS

```
int getpw (uid, buf)
int uid;
char *buf;
```

DESCRIPTION

getpw searches the password file for a user ID number that equals **uid**, copies the line of the password file in which **uid** was found into the array pointed to by **buf**, and returns 0. **getpw** returns non-zero if **uid** cannot be found.

This routine is included only for compatibility with prior systems and should not be used; see **getpwent**(3C) for routines to use instead.

FILES

/etc/passwd

SEE ALSO

getpwent(3C), passwd(4)

DIAGNOSTICS

getpw returns non-zero on error.

WARNINGS

The above routine uses **<stdio.h>** which causes it to increase, more than might be expected, the size of programs not otherwise using standard I/O.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent – get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent ( )

struct passwd *getpwuid (uid)
uid_t uid;

struct passwd *getpwnam (name)
char *name;

void setpwent ( )

void endpwent ( )

struct passwd *fgetpwent (f)
FILE *f;
```

DESCRIPTION

getpwent, getpwuid and **getpwnam** each returns a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/passwd* file. Each line in the file contains a "passwd" structure, declared in the **<pwd.h>** header file:

```
struct passwd {
        char    *pw_name;
        char    *pw_passwd;
        uid_t   pw_uid;
        gid_t   pw_gid;
        char    *pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

This structure is declared in **<pwd.h>** so it is not necessary to redeclare it.

The fields have meanings described in **passwd**(4).

getpwent when first called returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. **getpwuid** searches from the beginning of the file until a numerical user id matching **uid** is found and returns a pointer to the particular structure in which it was found. **getpwnam** searches from the beginning of the file until a login name matching **name** is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a **NULL** pointer.

A call to **setpwent** has the effect of rewinding the password file to allow repeated searches. **endpwent** may be called to close the password file when processing is complete.

fgetpwent returns a pointer to the next passwd structure in the stream **f**, which matches the format of */etc/passwd*.

FILES

/etc/passwd

SEE ALSO

mkpwdbm(1), getlogin(3C), getgrent(3C), passwd(4)

DIAGNOSTICS

A **NULL** pointer is returned on **EOF** or error.

WARNINGS

The above routines use *<stdio.h>* which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getut: getutent, getutid, getutline, pututline, setutent, endutent, utmpname, getutpid – access utmp file entry

SYNOPSIS

#include <utmp.h>

struct utmp *getutent ( )

struct utmp *getutid (id)
struct utmp *id;

struct utmp *getutline (line)
struct utmp *line;

void pututline (utmp)
struct utmp *utmp;

void setutent ( )

void endutent ( )

void utmpname (file)
char *file;

Sequent extensions:

struct utmp *getutpid (pid)
struct utmp *pid;

char *ut_find_host(line)
char *line;

in ut_get_entry(ut_entry, ut_host)
struct utmp **ut_entry
char **ut_host

DESCRIPTION

getutent, getutid , getutline and **getutpid** each return a pointer to a structure of the following type:

```
struct utmp {
  char          ut_user[8];     /* User login name */
  char          ut_id[4];       /* /etc/inittab ID (usually line #)*/
  char          ut_line[12];    /* device name (console, lnxx) */
  short         ut_pid;         /* process ID */
  short         ut_type;        /* type of entry */
  struct        exit_status {
    short       e_termination;  /* Process termination status */
    short       e_exit;         /* Process exit status */
  } ut_exit;                    /* The exit status of a process
                                 * marked as DEAD_PROCESS. */
  time_t        ut_time;        /* time entry was made */
};
```

getutent reads in the next entry from a *utmp*-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.

getutid searches the *utmp* file until it finds an entry with a **ut_type** matching **id->ut_type** if the type specified is **RUN_LVL, BOOT_TIME, OLD_TIME** or

NEW_TIME. If the type specified in **id** is **INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS** or **DEAD_PROCESS,** then **getutid** will return a pointer to the first entry whose type is one of these four and whose **ut_id** field matches **id->ut_id**. If the end of file is reached without a match, it fails.

getutline searches the *utmp* file until it finds an entry of the type **LOGIN_PROCESS** or **USER_PROCESS** which also has a **ut_line** string matching the **line->ut_line** string. If the end of file is reached without a match, it fails.

Pututline writes out the supplied **utmp** structure into the *utmp* file. It uses **getutid** to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of **pututline** will have searched for the proper entry using one of the **getut** routines. If so, **pututline** will not search. If **pututline** does not find a matching slot for the new entry, it will add a new entry to the end of the file.

setutent resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.

endutent closes the currently open file.

utmpname allows the user to change the name of the file examined, from */etc/utmp* to any other file. It is most often expected that this other file will be */etc/wtmp*. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. **utmpname** does not open the file. It just closes the old file if it is currently open and saves the new file name.

getutpid searches the *utmp* file for an entry of type **INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS,** or **DEAD_PROCESS** with a pid matching the pid in the structure passed to this routine. This procedure is a Sequent extension.

ut_find_host provides a way to determine the host machine associated with a tty line. **ut_find_host** returns a pointer to a static character string. If there is no name associated with the tty line, a (char *) NULL is returned. This procedure is a Sequent extension.

ut_get_entry provides a way to read the *utmp* file records and any host associated with a *utmp* line. The two pieces of information are guaranteed to be correct at the time of the read. **ut_get_entry** returns a non-zero value on error or end-of-file. Zero is returned if **ut_get_entry** is successful, and the first argument is set to point to a static *utmp* structure. The pointer is overwritten at each call. The second argument points to a static character buffer. If there is no host associated with this line, the first character of the string is NULL. This procedure is a Sequent extension. The following code can be used to walk through the *utmp* file.

```
struct utmp *u;
char *h;
int i = 1;

while (ut_get_entry(&u, &h) == 0 {
  if (u->ut_line[0] != 0)
    printf("%d: line = %s host = %s\n",
```

```
                    i, u->ut_line, h);
            i++;
        }
```

FILES

/etc/utmp
/etc/utmp_dbm.dir
/etc/utmp_dbm.pag
/etc/wtmp

SEE ALSO

ttyslot(3C), utadduser(3x), utmp(4)

DIAGNOSTICS

A NULL pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

WARNINGS

The above routine uses *<stdio.h>*, which causes the size of programs not otherwise using standard I/O to increase more than expected.

All information is saved in a static area, so it must be copied if it is to be saved.

NOTES

For performance reasons, the file */etc/utmp* is hashed. When accessing this file, a seek is done directly to the entry to be read. The routine **getutpid** is an extension, also provided for performance reasons and, if used, results in code that is not portable.

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made.

The following is true only if **utmpname** has been called to access a file other than */etc/utmp*. Calls to either **getutid** or **getutline** cause the routine to examine the static structure before performing more I/O. If the contents of the static structure match the search criteria, the routine stops searching. For this reason, to use **getutline** to search for multiple occurrences, it is necessary to zero out the static structure after each successful match, or **getutline** returns the same pointer over and over again. There is one exception to the rule about zeroing out the static structure before further reads are done. If the user has just modified the static structure and passed the pointer back to **pututline**, the implicit read done by **pututline** (if it finds that it is not already at the correct place in the file) does not change the contents of the static structure returned by the **getutent**, **getutid** or **getutline** routines.

These routines use buffered standard I/O for input, but **pututline** uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the *utmp* and *wtmp* files.

NAME

hsearch, hcreate, hdestroy – manage hash search tables

SYNOPSIS

```
#include <search.h>

ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;

int hcreate (nel)
unsigned nel;

void hdestroy ( )
```

DESCRIPTION

hsearch is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. **item** is a structure of type ENTRY (defined in the **<search.h>** header file) containing two pointers: **item.key** points to the comparison key, and **item.data** points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) **action** is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a **NULL** pointer.

hcreate allocates sufficient space for the table, and must be called before **hsearch** is used. **nel** is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

hdestroy destroys the search table, and may be followed by another call to **hcreate**.

NOTES

hsearch uses *open addressing* with a *multiplicative* hash function. However, its source code has many other options available which the user may select by compiling the **hsearch** source with the following symbols defined to the preprocessor:

DIV Use the *remainder modulo table size* as the hash function instead of the multiplicative algorithm.

USCR Use a User Supplied Comparison Routine for ascertaining table membership. The routine should be named **hcompar** and should behave in a mannner similar to **strcmp** (see **string**(3C)).

CHAINED Use a linked list to resolve collisions. If this option is selected, the following other options become available.

- **START** Place new entries at the beginning of the linked list (default is at the end).
- **SORTUP** Keep the linked list sorted by key in ascending order.

SORTDOWN
Keep the linked list sorted by key in descending order.

Additionally, there are preprocessor flags for obtaining debugging printout (**–DDEBUG**) and for including a test driver in the calling routine (**–DDRIVER**). The source code should be consulted for further details.

EXAMPLES

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```
#include <stdio.h>
#include <search.h>

struct info { /* this is the info stored in the table *
  int age, room;   /* other than the key. */
};
#define NUM_EMPL 5000 /* # of elements in search table */

main( )
{
   /* space to store strings */
   char string_space[NUM_EMPL*20];
   /* space to store employee info */
   struct info info_space[NUM_EMPL];
   /* next avail space in string_space */
   char *str_ptr = string_space;
   /* next avail space in info_space */
   struct info *info_ptr = info_space;
   ENTRY item, *found_item, *hsearch( );
   /* name to look for in table */
   char name_to_find[30];
   int i = 0;

   /* create table */
   (void) hcreate(NUM_EMPL);
   while (scanf("%s%d%d", str_ptr, &info_ptr->age,
            &info_ptr->room) != EOF && i++ < NUM_EMPL) {
      /* put info in structure, and structure in item */
      item.key = str_ptr;
      item.data = (char *)info_ptr;
      str_ptr += strlen(str_ptr) + 1;
      info_ptr++;
      /* put item into table */
      (void) hsearch(item, ENTER);
   }

   /* access table */
   item.key = name_to_find;
   while (scanf("%s", item.key) != EOF) {
         if ((found_item = hsearch(item, FIND)) != NULL) {
```

```
                /* if item is in the table */
                (void)printf("found %s, age = %d, room = %d\n",
                    found_item->key,
                    ((struct info *)found_item->data)->age,
                    ((struct info *)found_item->data)->room);
                    } else {
                (void)printf("no such employee %s\n",
                    name_to_find)
                    }
            }
        }
```

SEE ALSO

bsearch(3C), lsearch(3C), malloc(3C), malloc(3X), string(3C), tsearch(3C)

DIAGNOSTICS

hsearch returns a **NULL** pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

hcreate returns zero if it cannot allocate sufficient space for the table.

WARNINGS

hsearch and **hcreate** use **malloc**(3C) to allocate space.

Only one hash search table may be active at any given time.

NAME

isnan: isnand, isnanf – test for floating-point NaN (Not-A-Number)

SYNOPSIS

include <ieeefp.h>

int isnand (dsrc)
double dsrc;

int isnanf (fsrc)
float fsrc;

DESCRIPTION

isnand and **isnanf** return true (1) if the argument **dsrc** or **fsrc** is a NaN; otherwise they return false (0).

Neither routine generates any exception, even for signaling NaNs.

isnanf() is implemented as a macro included in *<ieeefp.h>*.

SEE ALSO

fpgetround(3C)

NAME

l3tol, ltol3 – convert between 3-byte integers and long integers

SYNOPSIS

```
void l3tol (lp, cp, n)
long *lp;
char *cp;
int n;

void ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;
```

DESCRIPTION

l3tol converts a list of **n** three-byte integers packed into a character string pointed to by **cp** into a list of long integers pointed to by **lp**.

ltol3 performs the reverse conversion from long integers (**lp**) to three-byte integers (**cp**).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

SEE ALSO

fs(4)

WARNINGS

Because of possible differences in byte ordering, the numerical values of the long integers are machine-dependent.

NAME

lockf – record locking on files

SYNOPSIS

```
#include <unistd.h>

int lockf (fildes, function, size)
long size;
int fildes, function;
```

DESCRIPTION

The **lockf** command allows sections of a file to be locked; advisory or mandatory write locks depending on the mode bits of the file (see **chmod**(2)). Locking calls from other processes which attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. (See **fcntl**(2) for more information about record locking.)

fildes is an open file descriptor. The file descriptor must have **O_WRONLY** or **O_RDWR** permission in order to establish lock with this function call.

function is a control value which specifies the action to be taken. The permissible values for **function** are defined in **<unistd.h>** as follows:

```
#define     F_ULOCK     0     /* Unlock a previously locked
                              section */
#define     F_LOCK      1     /* Lock a section for exclusive
                              use */
#define     F_TLOCK     2     /* Test and lock a section for
                              exclusive use */
#define     F_TEST      3     /* Test section for other
                              processes locks */
```

All other values of **function** are reserved for future extensions and will result in an error return if not implemented.

F_TEST is used to detect if a lock by another process is present on the specified section. **F_LOCK** and **F_TLOCK** both lock a section of a file if the section is available. **F_ULOCK** removes locks from a section of the file.

size is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If **size** is zero, the section from the current offset through the largest file offset is locked (i.e., from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked as such locks may exist past the end-of-file.

The sections locked with **F_LOCK** or **F_TLOCK** may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F_LOCK and **F_TLOCK** requests differ only by the action taken if the resource is not available. **F_LOCK** will cause the calling process to sleep until the resource is available. **F_TLOCK** will cause the function to return a –1 and set **errno** to [EACCES] error if the section is already locked by another process.

F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to **lockf** or **fcntl** scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The **alarm**(2) command may be used to provide a timeout facility in applications which require this facility.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and **errno** is set to indicate the error.

lockf fails if any of the following are true:

[EBADF]	**fildes** is not a valid open descriptor.
[EACCES]	**cmd** is **F_TLOCK** or **F_TEST** and the section is already locked by another process.
[EDEADLK]	**cmd** is **F_LOCK** and a deadlock would occur. Also the **cmd** is either **F_LOCK**, **F_TLOCK**, or **F_ULOCK**, and the number of entries in the lock table would exceed the number allocated on the system.
[ECOMM]	**fildes** is on a remote machine and the link to that machine is no longer active.

SEE ALSO

chmod(2), close(2), creat(2), fcntl(2), intro(2), open(2), read(2), write(2)

WARNINGS

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

Because in the future the variable **errno** will be set to EAGAIN rather than EACCES when a section of a file is already locked by another process, portable application programs should expect and test for either value.

NAME

lsearch, lfind – linear search and update

SYNOPSIS

```
#include <stdio.h>
#include <search.h>

char *lsearch (key, base, nelp, sizeof(*key), compar)
char *key;
char *base;
unsigned *nelp;
int (*compar)( );

char *lfind (key, base, nelp, sizeof(*key), compar)
char *key;
char *base;
unsigned *nelp;
int (*compar)( );
```

DESCRIPTION

lsearch is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. **key** points to the datum to be sought in the table. **base** points to the first element in the table. **nelp** points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. **compar** is the name of the comparison function which the user must supply (**strcmp**, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

lfind is the same as **lsearch** except that if the datum is not found, it is not added to the table. Instead, a **NULL** pointer is returned.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLES

This fragment reads in less than **TABSIZE** strings of length less than **ELSIZE** and stores them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

      char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
      unsigned nel = 0;
      int strcmp( );
```

```
. . .
while (fgets(line, ELSIZE, stdin) != NULL &&
     nel < TABSIZE)
       (void) lsearch(line, (char *)tab, &nel,
                ELSIZE, strcmp);
. . .
```

SEE ALSO

bsearch(3C), hsearch(3C), string(3C), tsearch(3C)

DIAGNOSTICS

If the searched for datum is found, both **lsearch** and **lfind** return a pointer to it. Otherwise, **lfind** returns **NULL** and **lsearch** returns a pointer to the newly added element.

BUGS

Undefined results can occur if there is not enough room in the table to add a new item.

NAME

malloc, free, realloc, calloc – main memory allocator

SYNOPSIS

```
char *malloc (size)
unsigned size;

void free (ptr)
char *ptr;

char *realloc (ptr, size)
char *ptr;
unsigned size;

char *calloc (nelem, elsize)
unsigned nelem, elsize;
```

DESCRIPTION

malloc and **free** provide a simple general-purpose memory allocation package. **malloc** returns a pointer to a block of at least **size** bytes suitably aligned for any use.

The argument to **free** is a pointer to a block previously allocated by **malloc**; after **free** is performed this space is made available for further allocation, but its contents are left undisturbed.

Undefined results occur if the space assigned by **malloc** is overrun or if some random number is handed to **free**.

malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, combining adjacent free blocks as it searches. It calls **sbrk** (see **brk**(2)) to get more memory from the system when there is no suitable space already free.

realloc changes the size of the block pointed to by **ptr** to **size** bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If no free block of **size** bytes is available in the storage arena, then **realloc** will ask **malloc** to enlarge the arena by **size** bytes and will then move the data to the new space.

realloc also works if **ptr** points to a block freed since the last call of **malloc**, **realloc**, or **calloc**; thus sequences of **free**, **malloc** and **realloc** can exploit the search strategy of **malloc** to do storage compaction.

calloc allocates space for an array of **nelem** elements of size **elsize**. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO

brk(2), malloc(3X)

DIAGNOSTICS

malloc, **realloc** and **calloc** return a **NULL** pointer if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When this happens, the block pointed to by **ptr** may be destroyed.

NOTES

Search time increases when many objects have been allocated; that is, if a program allocates but never frees, then each successive allocation takes longer. For an alternate, more flexible implementation, see **malloc**(3X).

NAME

memory: memccpy, memchr, memcmp, memcpy, memset – memory operations

SYNOPSIS

```
#include <memory.h>

char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr (s, c, n)
char *s;
int c, n;

int memcmp (s1, s2, n)
char *s1, *s2;
int n;

char *memcpy (s1, s2, n)
char *s1, *s2;
int n;

char *memset (s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

memccpy copies characters from memory area **s2** into **s1**, stopping after the first occurrence of character **c** has been copied, or after **n** characters have been copied, whichever comes first. It returns a pointer to the character after the copy of **c** in **s1**, or a **NULL** pointer if **c** was not found in the first **n** characters of **s2**.

memchr returns a pointer to the first occurrence of character **c** in the first **n** characters of memory area **s**, or a **NULL** pointer if **c** does not occur.

memcmp compares its arguments, looking at the first **n** characters only, and returns an integer less than, equal to, or greater than 0, according as **s1** is lexicographically less than, equal to, or greater than **s2**.

memcpy copies **n** characters from memory area **s2** to **s1**. It returns **s1**. Warning: **memcpy** does not check that **s1** and **s2** overlap, as such, **memcpy** is not a direct replacement for the **BSD bcopy** routine.

memset sets the first **n** characters in memory area **s** to the value of character **c**. It returns **s**.

For user convenience, all these functions are declared in the optional **<memory.h>** header file.

WARNINGS

memcmp is implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

mkfifo – make a FIFO special file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo (path,mode)
char *path;
mode_t mode;
```

DESCRIPTION

The **mkfifo** routine creates a new FIFO special file named by the pathname **path**. The file permission bits of the new FIFO are initialized from **mode**. The file permission bits of **mode** are modified by the process's file creation mask (see **umask**()). Bits in **mode** other than the file permission bits are ignored.

The FIFO's owner ID is set to the process's efective user ID. The FIFO's group ID is set to the group ID of the directory in which the FIFO is being created.

Upon successful completion a value of zero is returned. Otherwise, a value of -1 is returned, no FIFO is created, and **errno** is set to indicate the error.

If any of the following conditions occur, **mkfifo()** returns -1 and sets **errno** to the corresponding value:

[EACCES]	A component of the path prefix denies search permission.
[EEXIST]	The named file already exists.
[ENAMETOOLONG]	The length of the path string exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.
[ENOENT]	A component of the path prefix does not exist or the path argument points to an empty string.
[ENOSPC]	The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
[ENOTDIR]	A component of the path prefix is not a directory.
[EROFS]	The named file resides on a read-only file system.

SEE ALSO

chmod(1), exec(2), pipe(2), stat(5), umask(1)

NAME

mktemp – make a unique filename

SYNOPSIS

char *mktemp (template)
char *template;

DESCRIPTION

mktemp replaces the contents of the string pointed to by **template** by a unique filename, and returns the address of **template**. The string in **template** should look like a filename with six trailing **Xs**; **mktemp** will replace the **Xs** with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate an existing file.

SEE ALSO

getpid(2), tmpfile(3S), tmpnam(3S)

DIAGNOSTIC

mktemp will assign to **template** the **NULL** string if it cannot create a unique name.

WARNINGS

If called more than 17,576 times in a single process, this function starts recycling previously used names.

NAME

monitor – prepare execution profile

SYNOPSIS

```
#include <mon.h>

void monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
WORD *buffer;
int bufsize, nfunc;
```

DESCRIPTION

An executable program created by **cc –p** automatically includes calls for **monitor** with default parameters; **monitor** need not be called explicitly except to gain fine control over profiling.

monitor is an interface to **profil**(2). **lowpc** and **highpc** are the addresses of two functions; **buffer** is the address of a (user supplied) array of **bufsize** WORDs (defined in the **<mon.h>** header file). **monitor** arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of **lowpc** and the highest is just below **highpc**. **lowpc** may not equal 0 for this use of **monitor**. At most **nfunc** call counts can be kept; only calls of functions compiled with the profiling option **–p** of **cc**(1) are recorded.

For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use the following:

```
extern etext;
...
monitor ((int (*)())2, &etext, buf, bufsize, nfunc);
```

etext lies just above all the program text; see **end**(3C).

To stop execution monitoring and write the results, use the following;

```
monitor ((int (*)())0, 0, 0, 0, 0);
```

prof(1) can then be used to examine the results.

The name of the file written by **monitor** is controlled by the environment variable **PROFDIR**. If **PROFDIR** does not exist, *mon.out* is created in the current directory. If **PROFDIR** exists but has no value, **monitor** does not do any profiling and creates no output file. Otherwise, the value of **PROFDIR** is used as the name of the directory in which to create the output file. If **PROFDIR** is then the file written is *dirname/pid.mon.out* where **pid** is the program's process id. (When **monitor** is called automatically by compiling with **cc –p**, the file created is *dirname/pid.progname* where **progname** is the name of the program.)

FILES

mon.out

SEE ALSO

cc(1), prof(1), profil(2), end(3C)

BUGS

The *dirname/pid.mon.out* form does not work; the *dirname/pid.progname* form (automatically called with **cc -p**) does work.

NAME

nlist – get entries from name list

SYNOPSIS

```
#include <nlist.h>

int nlist (filename, nl)
char *filename;
struct nlist *nl;
```

DESCRIPTION

nlist examines the name list in the executable file whose name is pointed to by **filename**, and selectively extracts a list of values and puts them in the array of nlist structures pointed to by **nl**. The name list **nl** consists of an array of structures containing names of variables, types and values. The list is terminated with a null name; that is, a null string is in the name position of the structure. Each variable name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. The type field will be set to 0 unless the file was compiled with the –g option. If the name is not found, both entries are set to 0. See **a.out**(4) for a discussion of the symbol table structure.

This function is useful for examining the system name list kept in the file **/unix**. In this way programs can obtain system addresses that are up to date.

NOTES

The **<nlist.h>** header file is automatically included by **<a.out.h>** for compatability. However, if the only information needed from **<a.out.h>** is for use of **nlist**, then including **<a.out.h>** is discouraged. If **<a.out.h>** is included, the line `#undef n_name` may need to follow it.

SEE ALSO

a.out(4)

DIAGNOSTICS

All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

nlist returns –1 upon error; otherwise it returns 0.

NAME

perror, errno, sys_errlist, sys_nerr – system error messages

SYNOPSIS

```
void perror (s)
char *s;

extern int errno;

extern char *sys_errlist[ ];

extern int sys_nerr;
```

DESCRIPTION

perror produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string **s** is printed first, then a colon and a blank, then the message and a new-line. (However, if s="" the colon is not printed.) To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable **errno**, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the array of message strings **sys_errlist** is provided; **errno** can be used as an index into this table to get the message string without the new-line. **sys_nerr** is the number of messages in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2)

NAME

putenv – change or add value to environment

SYNOPSIS

int putenv (string)
char *string;

DESCRIPTION

string points to a string of the form **name=value**. **putenv** makes the value of the environment variable **name** equal to **value** by altering an existing variable or creating a new one. In either case, the string pointed to by **string** becomes part of the environment, so altering the string will change the environment. The space used by **string** is no longer used once a new string-defining **name** is passed to **putenv**.

SEE ALSO

exec(2), getenv(3C), malloc(3C), environ(5)

DIAGNOSTICS

putenv returns non-zero if it was unable to obtain enough space via **malloc** for an expanded environment, otherwise zero.

WARNINGS

putenv manipulates the environment pointed to by **environ**, and can be used in conjunction with **getenv**. However, **envp** (the third argument to **main**) is not changed.

This routine uses **malloc**(3C) to enlarge the environment.

After **putenv** is called, environmental variables are not in alphabetical order.

A potential error is to call **putenv** with an automatic variable as the argument, then exit the calling function while **string** is still part of the environment.

NAME

putpwent – write password file entry

SYNOPSIS

```
#include <pwd.h>

int putpwent (p, f)
struct passwd *p;
FILE *f;
```

DESCRIPTION

putpwent is the inverse of **getpwent**(3C). Given a pointer to a passwd structure created by **getpwent** (or **getpwuid** or **getpwnam**), **putpwent** writes a line on the stream **f**, which matches the format of */etc/passwd*.

SEE ALSO

mkpwdbm(1), getpwent(3C)

DIAGNOSTICS

putpwent returns non-zero if an error was detected during its operation, otherwise zero.

WARNINGS

The above routine uses *<stdio.h>*, which causes an increase in the size of programs, not otherwise using standard I/O, more than might be expected.

NAME

qsort – quicker sort

SYNOPSIS

void qsort ((char *) base, nel, sizeof (*base), compar)
unsigned nel;
int (*compar)( );

DESCRIPTION

qsort is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

base points to the element at the base of the table. **nel** is the number of elements in the table. **compar** is the name of the comparison function, which is called with two arguments that point to the elements being compared. As the function must return an integer less than, equal to, or greater than zero, so must the first argument to be considered be less than, equal to, or greater than the second.

NOTES

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The order in the output of two items which compare as equal is unpredictable.

SEE ALSO

sort(1), bsearch(3C), lsearch(3C), string(3C)

NAME

rand, srand – simple random-number generator

SYNOPSIS

int rand ( )

void srand (seed)
unsigned seed;

DESCRIPTION

rand uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudorandom numbers in the range from 0 to $2^{15}-1$.

srand can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTES

The spectral properties of **rand** are limited. **drand48**(3C) provides a much better, though more elaborate, random-number generator.

SEE ALSO

drand48(3C)

NAME

setjmp, longjmp – non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp (env)
jmp_buf env;

void longjmp (env, val)
jmp_buf env;
int val;
```

DESCRIPTION

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

setjmp saves its stack environment in **env** (whose type, is defined in the **<setjmp.h>** header file) for later use by **longjmp**. It returns the value 0.

longjmp restores the environment saved by the last call of **setjmp** with the corresponding **env** argument. After **longjmp** is completed, program execution continues as if the corresponding call of **setjmp** (which must not itself have returned in the interim) had just returned the value **val**. **longjmp** cannot cause **setjmp** to return the value 0. If **longjmp** is invoked with a second argument of 0, **setjmp** returns 1. At the time of the second return from **setjmp**, all external and static variables have values as of the time **longjmp** is called (see example). The values of register and automatic variables are undefined.

In a future release, C language users will be able to identify syntactically those automatic variables on whose values they need to rely after the second return from **setjmp**.

EXAMPLES

```
#include <setjmp.h>

jmp_buf env;
int i = 0;
main ()
{
  void exit();

  if(setjmp(env) != 0) {
    (void) printf("value of i on 2nd return from setjmp: %d\n", i);
    exit(0);
  }
  (void) printf("value of i on 1st return from setjmp: %d\n", i);
  i = 1;
  g();
  /*NOTREACHED*/
}

g()

{
```

```
    longjmp(env, 1);
    /*NOTREACHED*/
}
```

If the *a.out* resulting from this C language code is run, the output will be:

value of i on 1st return from setjmp:0

value of i on 2nd return from setjmp:1

SEE ALSO

signal(2), sigsetjmp(2)

WARNINGS

If **longjmp** is called even though **env** was never primed by a call to **setjmp**, or when the last such call was in a function that has since returned, absolute chaos is guaranteed.

NAME

setlocale – set program's location

SYNOPSIS

include <local.h>

char *setlocale (category, locale)
int category
char *locale

DESCRIPTION

The **setlocale** funciton selects the approprate portion of the program's locale as specified by the **category** and **locale** arguments. **setlocale** may be used to change or query the program's entire current locale or portions of it. A value of LC_ALL for **category** names the program's entire locale; other values for **category** name only a portion of the program's locale. LC_COLLATE affects the behavior of string collating functions. LC_CTYPE affects the behavior of character handling functions. LC_MONETARY affects monetary formatting information returned by various functions. LC_NUMERIC affects the decimal-point character for the formatted input-output functions and the string conversion functions. LC_TIME affects the behavior of **strftime**.

The C Standard allows the specification of an implementation-defined native environment for the **setlocale** function, which sets a specific category to an implementation-defined default. For this system, this corresponds to the value of the environment variables.

A specific category may be set to the value of its corresponding environment variable by passing the **locale** argument with a pointer to a null string. A value of "C" for **locale** specifies the minimal environment for C translation.

Possible values for **category** in the order they are set are as follows:

LC_CTYPE
LC_COLLATE
LC_TIME
LC_NUMERIC
LC_MONETARY

In all cases when the **locale** argument is specified as a pointer to a null string, **setlocale** first checks the value of the corresponding environment variable (for example, **LC_CTYPE** for the LC_CTYPE category). If it is valid, that is, if it points to the name of a valid locale), **setlocale** sets the specified category of the international environment to that value, and returns the string corresponding to the locale set, that is, the value of the environment variable, not "". If the value is invalid, **setlocale** returns a **NULL** pointer and the international environment is not changed.

If the environment variable corresponding to the specified category is not set, or is set to an empty string, **setlocale** returns a **NULL** pointer and the locale remains unchanged, unless the **LANG** environment variable is set. If **LANG** is valid, **setlocale** sets the category to the corresponding value of **LANG**. To set all categories in the international environment, **setlocale** is invoked as follows:

```
setlocale (LC_ALL,"");
```

setlocale first checks all environment variables. If any environment variable is invalid, **setlocale** returns a **NULL** pointer and the international environment is not changed. If all relevant environment variables are valid, **setlocale** sets the international environment to reflect the values of the environment variables.

Using this seheme, the categories corresponding to the environment variables override the value of the **LANG** environment variable for a particular category.

At program startup, the equivalent of the following is executed:

```
setlocale(LC_ALL, "C");
```

DIAGNOSTICS

If a pointer to a string is given for **locale** and the selection can be honored, **setlocale** returns the string associated with the specified category for the new locale. If the selection cannot be honored, **setlocale** returns a **NULL** pointer and the program's locale is not changed.

A **NULL** pointer for **locale** causes **setlocale** to return the string associated with a category for the program's current locale; the program's locale is not changed.

The string returned by **setlocale** is such that a subsequent call with that string and its associated category restores that part of the program's locale. The string returned shall not be modified by the program, but may be overwritten by a subsequent call to **setlocale**.

NAME

sigsetjmp, siglongjmp – non-local jumps

SYNOPSIS

```
#include <setjmp.h>

int sigsetjmp (env, savemask)
sigjmp_buf env;
int savemask;

void siglongjmp (env, val)
sigjmp_buf env;
int val;
```

DESCRIPTION

The **sigsetjmp** macro complies with the definition of the **setjmp** macro in the C Standard. If the value of **savemask** is not zero, **sigsetjmp** also saves the process's current signal mask (see *<signal.h>*) as part of the calling environment.

The **siglongjmp** function complies with the definition of the **longjmp** function in the C Standard. If and only if the **env** argument was initialized by a call to **sigsetjmp** with a non-zero **savemask** argument, **siglongjmp** restores the saved signal mask.

SEE ALSO

sigaction(2), sigprocmask(2), sigsuspend(2)

NAME

sleep – suspend execution for interval

SYNOPSIS

unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of **seconds** specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) because any caught signal will terminate the **sleep** following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by **sleep** will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested **sleep** time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling **sleep**. If the **sleep** time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the **sleep** routine returns. But if the **sleep** time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening **sleep**.

SEE ALSO

alarm(2), pause(2), signal(2)

NAME

ssignal, gsignal – software signals

SYNOPSIS

```
#include <signal.h>

int (*ssignal (sig, action))( )
int sig, (*action)( );

int gsignal (sig)
int sig;
```

DESCRIPTION

ssignal and **gsignal** implement a software facility similar to **signal**(2). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions, and is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the range 1 through 16 inclusive. A call to **ssignal** associates a procedure, **action**, with the software signal **sig**; the software signal, **sig**, is raised by a call to **gsignal**. Raising a software signal causes the action established for that signal to be **taken**.

The first argument to **ssignal** is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) **action function** or one of the manifest constants **SIG_DFL** (default) or **SIG_IGN** (ignore). **ssignal** returns the action previously established for that signal type; if no action has been established or the signal number is illegal, **ssignal** returns **SIG_DFL**.

gsignal raises the signal identified by its argument, **sig**:

If an action function has been established for **sig**, then that action is reset to **SIG_DFL** and the action function is entered with argument **sig**. **gsignal** returns the value returned to it by the action function.

If the action for **sig** is **SIG_IGN**, **gsignal** returns the value 1 and takes no other action.

If the action for **sig** is **SIG_DFL**, **gsignal** returns the value 0 and takes no other action.

If **sig** has an illegal value or no action was ever specified for **sig**, **gsignal** returns the value 0 and takes no other action.

SEE ALSO

signal(2), sigset(2), sigaction(2)

NOTES

There are some additional signals with numbers outside the range 1 through 16 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 16 are legal, although their use may interfere with the operation of the Standard C Library.

NAME

stdipc: ftok – standard interprocess communication package

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(path, id)
char *path;
char id;
```

DESCRIPTION

All interprocess communication facilities require the user to supply a key to be used by the **msgget**(2), **semget**(2), and **shmget**(2) system calls to obtain interprocess communication identifiers. One suggested method for forming a key is to use the **ftok** subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it is possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

ftok returns a key based on **path** and **id** that is usable in subsequent **msgget**, **semget**, and **shmget** system calls. **path** must be the pathname of an existing file that is accessible to the process. **id** is a character which uniquely identifies a project. Note that **ftok** returns the same key for linked files when called with the same **id** and that it returns different keys when called with the same filename but different **ids**.

SEE ALSO

intro(2), msgget(2), semget(2), shmget(2)

DIAGNOSTICS

ftok returns **(key_t) –1** if **path** does not exist or if it is not accessible to the process.

WARNINGS

If the file whose **path** is passed to **ftok** is removed when keys still refer to the file, future calls to **ftok** with the same **path** and **id** return an error. If the same file is recreated, then **ftok** is likely to return a different key than it did the original time it was called.

NAME

strftime – format time information

SYNOPSIS

```
#include <time.h>
int strftime (s, maxsize, format, tm)
char *s;
char *format;
struct tm *timeptr;
```

DESCRIPTION

The **strftime** function places characters into the array pointed to by **s** as controlled by the string pointed to by **format.** The **format** string is a multibyte character sequence, beginning and ending in its initial shift state. The **format** string consists of zero or more conversion specifications and ordinary multibyte characters. A conversion specification consists of a % character followed by a charater that determines the conversion specification's behavior. All ordinary multibyte characters (including the terminating **NULL** character) are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than **maxsize** characters are placed into the array. Each conversion specification is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the program's locale and by the values contained in the structure pointed to by **timeptr.** If a conversion specification is not one of these, the behavior is undefined.

%a is replaced by the locale's abbreviated weekday name.

%A is replaced by the locale's full weekday name.

%b is replaced bby the locale's abbreviated month name.

%B is replaced by the locale's full month name.

%c is replaced by the locale's appropriate date and time representation.

%d is replaced by the day of the month as a decimal number (01-31).

%H is replaced by the hour (24-hour clock) as a decimal number (00-23).

%I is replaced by the hour (12-hour clock) as a decimal number (01-12).

%j is replaced by the day of the year as a decimal number (001-366).

%m is replaced by the month as a decimal number (01-12).

%M is replaced by the minute as a decimal number (00-59).

%P is replaced by the locale's equivalent of either AM or PM.

%S is replaced by the second as a decimal number (00-60).

%U is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number (00-53).

%w is replaced by the weekday as a decimal number [0 (Sunday)-6].

%W is replaced by the week number of the year (Monday as the first of the week) as a decimal number (00-53).

%x is replaced by the locale's appropriate date representation.

%X is replaced by the locale's appropriate time representation.

%y is replaced by the year without century as a decimal number (00-99).

%Y is replaced by the year with century as a decimal number.

%Z is replaced by the time zone name, or by no characters if no time zone is determinable.

%% is replaced by %

DIAGNOSTICS

If the total number of resulting characters, including the terminating NULL character, is not more than **maxsize, strftime** returns the number of characters placed into the array pointed to by **s**, not including the terminating NULL charcter. Otherwise, zero is returned and the contents of the array are indetermininate.

NAME

string: strcat, strdup, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok – string operations

SYNOPSIS

```
#include <string.h>
#include <sys/types.h>

char *strcat (s1, s2)
char *s1, *s2;

char *strdup (s1)
char *s1;

char *strncat (s1, s2, n)
char *s1, *s2;
size_t n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
size_t n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
size_t n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s;
int c;

char *strrchr (s, c)
char *s;
int c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;
```

DESCRIPTION

The arguments **s1**, **s2** and **s** point to strings (arrays of characters terminated by a NULL character). The functions **strcat**, **strncat**, **strcpy**, and all alter **s1**. These functions do not check for overflow of the array pointed to by **s1**.

strcat appends a copy of string **s2** to the end of string **s1**.

strdup returns a pointer to a new string which is a duplicate of the string pointed to by **s1**. The space for the new string is obtained using **malloc**(3C). If the new string can not be created, **NULL** is returned.

strncat appends at most **n** characters. Each returns a pointer to the NULL–terminated result.

strcmp compares its arguments and returns an integer less than, equal to, or greater than 0, according as **s1** is lexicographically less than, equal to, or greater than **s2**. **strncmp** makes the same comparison but looks at at most **n** characters.

strcpy copies string **s2** to **s1**, stopping after the NULL character has been copied. **strncpy** copies exactly **n** characters, truncating **s2** or adding NULL characters to **s1** if necessary. The result is not NULL–terminated if the length of **s2** is **n** or more. Each function returns **s1**.

strlen returns the number of characters in **s**, not including the terminating NULL character.

strchr (**strrchr**) returns a pointer to the first (last) occurrence of character **c** in string **s**, or a NULL pointer if **c** does not occur in the string. The NULL character terminating a string is considered to be part of the string.

strpbrk returns a pointer to the first occurrence in string **s1** of any character from string **s2**, or a NULL pointer if no character from **s2** exists in **s1**.

strspn (**strcspn**) returns the length of the initial segment of string **s1** which consists entirely of characters from (not from) string **s2**.

Strtok considers the string **s1** to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string **s2**. The first call (with pointer **s1** specified) returns a pointer to the first character of the first token, and will have written a NULL character into **s1** immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string **s1** immediately following that token. In this way subsequent calls will work through the string **s1** until no tokens remain. The separator string **s2** may be different from call to call. When no token remains in **s1**, a NULL pointer is returned.

For user convenience, all these functions are declared in the optional **<string.h>** header file.

SEE ALSO

malloc(3C), malloc(3X)

WARNINGS

strcmp and **strncmp** are implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high-order bit set not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

strstr – find first occurrence of character

SYNOPSIS

```
#include <string.h>
char *strstr (s1, s2);
char *s1;
char *s2;
```

DESCRIPTION

The **strstr** function locates the first occurrence in the string pointed to by **s1** of the sequence of characters (excluding the terminating null character) in the string pointed to by **s2.**

DIAGNOSTICS

The **strstr** function returns a pointer to the located string, or a **NULL** pointer if the string is not found. If **s2** points to a string with zero length, the function returns **s1.**

NAME

strtod, atof – convert string to double-precision number

SYNOPSIS

double strtod (str, ptr)
char *str, **ptr;

double atof (str)
char *str;

DESCRIPTION

strtod returns as a double-precision floating-point number the value represented by the character string pointed to by **str**. The string is scanned up to the first unrecognized character.

strtod recognizes an optional string of white-space characters (as defined by **isspace** in **ctype**(3C)), then an optional sign, then a string of digits optionally containing a decimal point, then an optional **e** or **E** followed by an optional sign or space, followed by an integer.

If the value of **ptr** is not **(char **)NULL**, a pointer to the character terminating the scan is returned in the location pointed to by **ptr**. If no number can be formed, ***ptr** is set to **str**, and zero is returned.

atof(str) is equivalent to **strtod(str, (char **)NULL)**.

SEE ALSO

ctype(3C), scanf(3S), strtol(3C)

DIAGNOSTICS

If the correct value would cause overflow, ±HUGE (as defined in **<math.h>**) is returned (according to the sign of the value), and **errno** is set to ERANGE. If the correct value would cause underflow, zero is returned and **errno** is set to ERANGE.

NAME

strtol, atol, atoi – convert string to integer

SYNOPSIS

long strtol (str, ptr, base)
char *str, **ptr;
int base;

long atol (str)
char *str;

int atoi (str)
char *str;

DESCRIPTION

strtol returns as a long integer the value represented by the character string pointed to by **str**. The string is scanned up to the first character inconsistent with the base. Leading white-space characters (as defined by **isspace** in **ctype**(3C)) are ignored.

If the value of **ptr** is not **(char **)NULL**, a pointer to the character terminating the scan is returned in the location pointed to by **ptr**. If no integer can be formed, that location is set to **str**, and zero is returned.

If **base** is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "0X" is ignored if **base** is 16.

If **base** is zero, the string itself determines the base thusly: After an optional leading sign a leading zero indicates octal conversion, and a leading "0x" or "0X" hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

atol(str) is equivalent to **strtol(str, (char **)NULL, 10)**.

atoi(str) is equivalent to **(int) strtol(str, (char **)NULL, 10)**.

SEE ALSO

ctype(3C), scanf(3S), strtod(3C)

WARNINGS

Overflow conditions are ignored.

NAME

swab – swap bytes

SYNOPSIS

void swab (from, to, nbytes)
char *from, *to;
int nbytes;

DESCRIPTION

swab copies **nbytes** bytes pointed to by **from** to the array pointed to by **to**, exchanging adjacent even and odd bytes. **nbytes** should be even and non-negative. If **nbytes** is odd and positive **swab** uses **nbytes**–1 instead. If **nbytes** is negative, **swab** does nothing.

NAME

tsearch, tfind, tdelete, twalk – manage binary search trees

SYNOPSIS

```
#include <search.h>

char *tsearch (key, rootp, compar)
char *key;
char *rootp;
int (*compar)( );

char *tfind (key, rootp, compar)
char *key;
char *rootp;
int (*compar)( );

char *tdelete (key, rootp, compar)
char *key;
char *rootp;
int (*compar)( );

void twalk (root, action)
char *root;
void (*action)( );
```

DESCRIPTION

tsearch, tfind, tdelete, and **twalk** are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

tsearch is used to build and access the tree. **key** is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to ***key** (the value pointed to by **key**), a pointer to this found datum is returned. Otherwise, ***key** is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. **rootp** points to a variable that points to the root of the tree. A **NULL** value for the variable pointed to by **rootp** denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like **tsearch, tfind** searches for a datum in the tree, returning a pointer to it if found. However, if it is not found, **tfind** will return a **NULL** pointer. The arguments for **tfind** are the same as for **tsearch.**

tdelete deletes a node from a binary search tree. The arguments are the same as for **tsearch.** The variable pointed to by **rootp** will be changed if the deleted node was the root of the tree. **tdelete** returns a pointer to the parent of the deleted node, or a **NULL** pointer if the node is not found.

twalk traverses a binary search tree. **root** is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) **action** is the name of a routine to be invoked at each node. This

routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type **typedef enum { preorder, postorder, endorder, leaf } VISIT;** (defined in the **<search.h>** header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLES

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>

struct node {                   /* pointers to these are */
                                /* stored in the tree */
  char *string;
  int length;
};
char string_space[10000];
                                /* space to store strings */
struct node nodes[500];         /* nodes to store */
struct node *root = NULL;
                                /* this points to the root */

main( )
{
  char *strptr = string_space;
  struct node *nodeptr = nodes;
  void print_node( ), twalk( );
  int i = 0, node_compare( );

  while (gets(strptr) != NULL && i++ < 500)  {
                                /* set node */
    nodeptr->string = strptr;
    nodeptr->length = strlen(strptr);
                                /* put node into the tree */
    (void) tsearch((char *)nodeptr, (char **) &root,
        node_compare);
                                /* adjust pointers, so we */
                                /* don't overwrite tree */
    strptr += nodeptr->length + 1;
    nodeptr++;
  }
  twalk((char *)root, print_node);
```

```
}
/*
        This routine compares two nodes, based on an
        alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
char *node1, *node2;
{
        return strcmp(((struct node *)node1)->string,
        ((struct node *) node2)->string);
}
/*
        This routine prints out a node, the first time
        twalk encounters it.
*/
void
print_node(node, order, level)
char **node;
VISIT order;
int level;
{
        if (order == preorder || order == leaf)  {
                (void)printf("string = %20s,  length = %d\n",
                        (*((struct node **)node))->string,
                        (*((struct node **)node))->length);
        }
}
```

SEE ALSO

bsearch(3C), hsearch(3C), lsearch(3C)

DIAGNOSTICS

A **NULL** pointer is returned by **tsearch** if there is not enough space available to create a new node.

A **NULL** pointer is returned by **tfind** and **tdelete** if **rootp** is **NULL** on entry. If the datum is found, both **tsearch** and **tfind** return a pointer to it. If not, **tfind** returns **NULL**, and **tsearch** returns a pointer to the inserted item.

WARNINGS

The **root** argument to **twalk** is one level of indirection less than the **rootp** arguments to **tsearch** and **tdelete**.

There are two nomenclatures used to refer to the order in which tree nodes are visited. **tsearch** uses preorder, postorder and endorder to respectively refer to visting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

If the calling function alters the pointer to the root, results are unpredictable.

NAME

ttyname, isatty – find name of a terminal

SYNOPSIS

char *ttyname (fildes)
int fildes;

int isatty (fildes)
int fildes;

DESCRIPTION

ttyname returns a pointer to a string containing the **NULL**–terminated path name of the terminal device associated with file descriptor **fildes.**

First **ttyname** calls the **dev_to_ttyname** procedure to rapidly generate a potiental pathname for the terminal. If the device node for the path returned matches the device specified by **fildes** the pathname is returned immediately. If the device node does not match, **ttyname** will search in the /dev directory until a match is found.

isatty returns 1 if **fildes** is associated with a terminal device, 0 otherwise.

FILES

/dev/* /dev/tty*

DIAGNOSTICS

ttyname returns a **NULL** pointer if **fildes** does not describe a terminal device in directory */dev*.

WARNINGS

The return value points to static data whose content is overwritten by each call.

SEE ALSO

dev_to_ttyname(3C)

NAME

ttyslot – find the slot in the utmp file of the current user

SYNOPSIS

int ttyslot ( )

DESCRIPTION

ttyslot returns the index of the current user's entry in the */etc/utmp* file. This is accomplished by actually scanning the file */etc/inittab* for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

FILES

/etc/inittab
/etc/utmp

SEE ALSO

getut(3C), ttyname(3C)

DIAGNOSTICS

A value of -1 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

NAME

bessel: j0, j1, jn, y0, y1, yn – Bessel functions

SYNOPSIS

```
#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x)
int n;
double x;

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;
```

DESCRIPTION

j0 and **j1** return Bessel functions of **x** of the first kind of orders 0 and 1 respectively. **jn** returns the Bessel function of **x** of the first kind of order **n**.

y0 and **y1** return Bessel functions of **x** of the second kind of orders 0 and 1 respectively. **yn** returns the Bessel function of **x** of the second kind of order **n**. The value of **x** must be positive.

SEE ALSO

matherr(3M)

DIAGNOSTICS

Non-positive arguments cause **y0**, **y1** and **yn** to return the value -**HUGE** and to set **errno** to **EDOM**. In addition, a message indicating **DOMAIN** error is printed on the standard error output.

Arguments too large in magnitude cause **j0**, **j1**, **y0** and **y1** to return zero and to set **errno** to **ERANGE**. In addition, a message indicating TLOSS error is printed on the standard error output.

These error-handling procedures may be changed with the function **matherr**(3M).

NAME

erf, erfc – error function and complementary error function

SYNOPSIS

```
#include <math.h>

double erf (x)
double x;

double erfc (x)
double x;
```

DESCRIPTION

erf returns the error function of **x**, defined as $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

erfc, which returns 1.0 – **erf(x)**, is provided because of the extreme loss of relative accuracy if **erf(x)** is called for large **x** and the result subtracted from 1.0 (e.g., for **x** = 5, 12 places are lost).

SEE ALSO

exp(3M)

NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root functions

SYNOPSIS

```
#include <math.h>

double exp (x)
double x;

double log (x)
double x;

double log10 (x)
double x;

double pow (x, y)
double x, y;

double sqrt (x)
double x;
```

DESCRIPTION

exp returns e^{x}.

log returns the natural logarithm of **x**. The value of **x** must be positive.

log10 returns the logarithm base ten of **x**. The value of **x** must be positive.

pow returns x^{y}. If **x** is zero, **y** must be positive. If **x** is negative, **y** must be an integer.

sqrt returns the non-negative square root of **x**. The value of **x** may not be negative.

SEE ALSO

hypot(3M), matherr(3M), sinh(3M).

DIAGNOSTICS

exp returns **HUGE** when the correct value would overflow, or 0 when the correct value would underflow, and sets **errno** to **ERANGE**.

log and **log10** return **–HUGE** and set **errno** to **EDOM** when **x** is non-positive. A message indicating **DOMAIN** error (or **SING** error when **x** is 0) is printed on the standard error output.

pow returns 0 and sets **errno** to **EDOM** when **x** is 0 and **y** is non-positive, or when **x** is negative and **y** is not an integer. In these cases a message indicating DOMAIN error is printed on the standard error output. When the correct value for **pow** would overflow or underflow, **pow** returns **±HUGE** or 0 respectively, and sets to **ERANGE**.

sqrt returns 0 and sets **errno** to **EDOM** when **x** is negative. A message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function **matherr**(3M).

NAME

floor, ceil, fmod, fabs – floor, ceiling, remainder, absolute value functions

SYNOPSIS

```
#include <math.h>

double floor (x)
double x;

double ceil (x)
double x;

double fmod (x, y)
double x, y;

double fabs (x)
double x;
```

DESCRIPTION

floor returns the largest integer (as a double-precision number) not greater than **x**.

ceil returns the smallest integer not less than **x**.

fmod returns the floating-point remainder of the division of **x** by **y**: **x** if **y** is zero or if **x/y** would overflow; otherwise the number **f** with the same sign as **x**, such that **x** = **iy** + **f** for some integer **i**, and $|f| < |y|$.

fabs returns the absolute value of **x**, $|x|$.

SEE ALSO

abs(3C)

NAME

gamma – log gamma function

SYNOPSIS

#include <math.h>

double gamma (x)
double x;

extern int signgam;

DESCRIPTION

gamma returns ln(| $\Gamma(x)$ |), where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t}t^{x-1}dt$. The sign of $\Gamma(x)$ is returned in the external integer **signgam**. The argument **x** may not be a non-positive integer.

The following C program fragment might be used to calculate Γ:

```
if ((y = gamma(x)) > LN_MAXDOUBLE)
      error();
y = signgam * exp(y);
```

where **LN_MAXDOUBLE** is the least value that causes **exp**(3M) to return a range error, and is defined in the **<values.h>** header file.

SEE ALSO

exp(3M), matherr(3M), values(5)

DIAGNOSTICS

For non-negative integer arguments **HUGE** is returned, and **errno** is set to **EDOM**. A message indicating SING error is printed on the standard error output.

If the correct value would overflow, **gamma** returns **HUGE** and sets **errno** to **ERANGE**.

These error-handling procedures may be changed with the function **matherr**(3M).

NAME

hypot – Euclidean distance function

SYNOPSIS

#include <math.h>

double hypot (x, y)
double x, y;

DESCRIPTION

hypot returns

sqrt(x * x + y * y),

taking precautions against unwarranted overflows.

SEE ALSO

matherr(3M)

DIAGNOSTICS

When the correct value would overflow, **hypot** returns **HUGE** and sets **errno** to **ERANGE.**

These error-handling procedures may be changed with the function **matherr**(3M).

NAME

matherr – error-handling function

SYNOPSIS

#include <math.h>

int matherr (x)
struct exception *x;

DESCRIPTION

matherr is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors, by including a function named **matherr** in their programs. **matherr** must be of the form described above. When an error occurs, a pointer to the exception structure **x** will be passed to the user-supplied **matherr** function. This structure, which is defined in the **<math.h>** header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1, arg2, retval;
};
```

The element **type** is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

DOMAIN	argument domain error
SING	argument singularity
OVERFLOW	overflow range error
UNDERFLOW	underflow range error
TLOSS	total loss of significance
PLOSS	partial loss of significance

The element **name** points to a string containing the name of the function that incurred the error. The variables **arg1** and **arg2** are the arguments with which the function was invoked. **retval** is set to the default value that will be returned by the function unless the user's **matherr** sets it to a different value.

If the user's **matherr** function returns non-zero, no error message will be printed, and **errno** will not be set.

If **matherr** is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized below. In every case, **errno** is set to EDOM or ERANGE and the program continues.

EXAMPLES

```
#include <math.h>

int
matherr(x)
register struct exception *x;
{
  switch (x->type) {
  case DOMAIN:
                        /* change sqrt to return  sqrt(-arg1), */
                        /* not 0 */
```

```
    if (!strcmp(x->name, "sqrt")) {
      x->retval = sqrt(-x->arg1);
      return (0);       /* print message and set errno */
    }
  case SING:
                        /* all other domain or sing errors, */
                        /* print message and abort */
    fprintf(stderr, "domain error in %s\n", x->name);
    abort( );
  case PLOSS:
                        /* print detailed error message */
    fprintf(stderr, "loss of significance in %s(%g) = %g\n",
      x->name, x->arg1, x->retval);
    return (1);         /* take no other action */
  }
  return (0);           /* all other errors, execute */
                        /* default procedure */
}
```

DEFAULT ERROR HANDLING PROCEDURES

BESSEL:	**TLOSS** [ERANGE] returns a message or 0 **PLOSS** [ERANGE] returns as much of the value as possible
y0, y1, yn (arg <= 0)	**DOMAIN** [EDOM] returns a message or –HUGE
EXP:	**OVERFLOW** [ERANGE] returns HUGE **UNDERFLOW** [ERANGE] returns 0
LOG, LOG10: (a < 0) (arg = 0)	 **DOMAIN** [EDOM] returns a message or –HUGE **SING** [EDOM] returns a message of –HUGE
POW:	**OVERFLOW** [ERANGE] returns HUGE or –HUGE **UNDERFLOW** [ERANGE] returns 0
neg ** non-int	 **DOMAIN** [EDOM] returns a message or 0
SQRT:	**DOMAIN** [EDOM] return a message or 0
GAMMA:	**SING** [EDOM] returns a message or HUGE
HYPOT:	**OVERFLOW** [ERANGE] returns HUGE
SINH:	**OVERFLOW** [ERANGE] returns HUGE or –HUGE

COSH:	**OVERFLOW** [ERANGE] returns **HUGE**
SIN, COS, TAN:	**UNDERFLOW** [ERANGE] returns a message or 0 **TLOSS** returns as much of the value as possible
ASIN, ACOS, ATAN2:	**DOMAIN** [EDOM] returns a message or 0

NAME

sinh, cosh, tanh – hyperbolic functions

SYNOPSIS

```
#include <math.h>

double sinh (x)
double x;

double cosh (x)
double x;

double tanh (x)
double x;
```

DESCRIPTION

sinh, **cosh**, and **tanh** return, respectively, the hyberbolic sine, cosine and tangent of their argument.

SEE ALSO

matherr(3M)

DIAGNOSTICS

sinh and **cosh** return **HUGE** (and **sinh** may return **-HUGE** for negative **x**) when the correct value would overflow and set **errno** to **ERANGE.**

These error-handling procedures may be changed with the function **matherr**(3M).

NAME

trig: sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS

```
#include <math.h>

double sin (x)
double x;

double cos (x)
double x;

double tan (x)
double x;

double asin (x)
double x;

double acos (x)
double x;

double atan (x)
double x;

double atan2 (y, x)
double y, x;
```

DESCRIPTION

sin, **cos** and **tan** return respectively the sine, cosine and tangent of their argument, **x**, measured in radians.

asin returns the arcsine of **x**, in the range $[-\pi/2,\pi/2]$.

acos returns the arccosine of **x**, in the range $[0,\pi]$.

atan returns the arctangent of **x**, in the range $[-\pi/2,\pi/2]$.

atan2 returns the arctangent of **y/x**, in the range $[-\pi,\pi]$, using the signs of both arguments to determine the quadrant of the return value.

SEE ALSO

matherr(3M)

DIAGNOSTICS

sin, **cos**, and **tan** lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return zero when there would otherwise be a complete loss of significance. In this case a message indicating **TLOSS** error is printed on the standard error output. For less extreme arguments causing partial loss of significance, a **PLOSS** error is generated but no message is printed. In both cases, **errno** is set to **ERANGE**.

If the magnitude of the argument of **asin** or **acos** is greater than one, or if both arguments of **atan2** are zero, zero is returned and **errno** is set to **EDOM**. In addition, a message indicating **DOMAIN** error is printed on the standard error output.

These error-handling procedures may be changed with the function **matherr**(3M).

NAME

t_accept – accept a connect request

SYNOPSIS

```
#include <tiuser.h>

int t_accept(fd, resfd, call)
int fd;
int resfd;
struct t_call *call;
```

DESCRIPTION

This function is issued by a transport user to accept a connect request. **fd** identifies the local transport endpoint where the connect indication arrived, **resfd** specifies the local transport endpoint where the connection is to be established, and **call** contains information required by the transport provider to complete the connection. **call** points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

netbuf is described in **intro**(3). In **call, addr** is the address of the caller, **opt** indicates any protocol-specific parameters associated with the connection, **udata** points to any user data to be returned to the caller, and **sequence** is the value returned by **t_listen** that uniquely associates the response with a previously received connect indication. Some transport providers may ignore the **addr** part of **call.**

A transport user may accept a connection on either the same, or on a different, local transport endpoint than the one on which the connect indication arrived. If the same endpoint is specified (i.e., **resfd=fd**), the connection can be accepted unless the following condition is true: The user has received other indications on that endpoint but has not responded to them (with **t_accept** or **t_snddis**). For this condition, **t_accept** will fail and set **t_errno** to TBADF.

If a different transport endpoint is specified (**resfd!=fd**), the endpoint must be bound to a protocol address and must be in the **T_IDLE** state [see **t_getstate**(3N)] before the **t_accept** is issued.

For both types of endpoints, **t_accept** will fail and set **t_errno** to TLOOK if there are indications (e.g., a connect or disconnect) waiting to be received on that endpoint.

The values of parameters specified by **opt** and the syntax of those values are protocol specific. The **udata** argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider as returned by **t_open** or **t_getinfo.** If the **len** [see **netbuf** in **intro**(3)] field of **udata** is zero, no data will be sent to the caller.

On failure, **t_errno** may be set to one of the following:

[TACCES]	The user does not have permission to accept a connection on the responding transport endpoint or use the specified options.
	endpoint referred to by **resfd** is not in the T_IDLE state.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_connect(3N), t_getstate(3N), t_listen(3N), t_open(3N), t_rcvconnect(3N)

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and **t_errno** is set to indicate the error.

NAME

t_alloc – allocate a library structure

SYNOPSIS

```
#include <tiuser.h>

char *t_alloc(fd, struct_type, fields)
int fd;
int struct_type;
int fields;
```

DESCRIPTION

The **t_alloc** function dynamically allocates memory for the various transport function argument structures as specified below. This function will allocate memory for the specified structure, and will also allocate memory for buffers referenced by the structure.

The structure to allocate is specified by **struct_type**, and can be one of the following:

T_BIND	**struct t_bind**
T_CALL	**struct t_call**
T_OPTMGMT	**struct t_optmgmt**
T_DIS	**struct t_discon**
T_UNITDATA	**struct t_unitdata**
T_UDERROR	**struct t_uderr**
T_INFO	**struct t_info**

where each of these structures may subsequently be used as an argument to one or more transport functions.

Each of the above structures, except **T_INFO**, contains at least one field of type **struct netbuf**. **netbuf** is described in **intro**(3). For each field of this type, the user may specify that the buffer for that field should be allocated as well. The **fields** argument specifies this option, where the argument is the bitwise-OR of any of the following:

T_ADDR The **addr** field of the **t_bind**, **t_call**, **t_unitdata**, or **t_uderr** structures.

T_OPT The **opt** field of the **t_optmgmt**, **t_call**, **t_unitdata**, or **t_uderr** structures.

T_UDATA The **udata** field of the **t_call**, **t_discon**, or **t_unitdata** structures.

T_ALL All relevant fields of the given structure.

For each field specified in **fields**, **t_alloc** will allocate memory for the buffer associated with the field, and initialize the **buf** pointer and **maxlen** [see **netbuf** in **intro**(3) for description of **buf** and **maxlen**] field accordingly. The length of the buffer allocated will be based on the same size information that is returned to the user on **t_open** and **t_getinfo**. Thus, **fd** must refer to the transport endpoint through which the newly allocated structure will be passed, so that the appropriate size information can be accessed. If the size value associated with any specified field is -1 or -2 (see **t_open** or **t_getinfo**), **t_alloc** will be unable to determine the size of the buffer to allocate and will

fail, setting **t_errno** to **TSYSERR** and **errno** to EINVAL. For any field not specified in **fields, buf** will be set to NULL and **maxlen** will be set to zero.

Use of **t_alloc** to allocate structures will help ensure the compatibility of user programs with future releases of the transport interface.

On failure, **t_errno** may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_free(3N), t_getinfo(3N), t_open(3N)

DIAGNOSTICS

On successful completion, **t_alloc** returns a pointer to the newly allocated structure. On failure, NULL is returned.

NAME

t_bind – bind an address to a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_bind(fd, req, ret)
int fd;
struct t_bind *req;
struct t_bind *ret;

DESCRIPTION

This function associates a protocol address with the transport endpoint specified by **fd** and activates that transport endpoint. In connection mode, the transport provider may begin accepting or requesting connections on the transport endpoint. In connectionless mode, the transport user may send or receive data units through the transport endpoint.

The **req** and **ret** arguments point to a **t_bind** structure containing the following members:

```
struct netbuf addr;
unsigned qlen;
```

netbuf is described in **intro**(3). The **addr** field of the **t_bind** structure specifies a protocol address and the **qlen** field is used to indicate the maximum number of outstanding connect indications.

req is used to request that an address, represented by the **netbuf** structure, be bound to the given transport endpoint. **len** [see **netbuf** in **intro**(3); also for **buf** and **maxlen**] specifies the number of bytes in the address and **buf** points to the address buffer. **maxlen** has no meaning for the **req** argument. On return, **ret** contains the address that the transport provider actually bound to the transport endpoint; this may be different from the address specified by the user in **req**. In **ret**, the user specifies **maxlen** which is the maximum size of the address buffer and **buf** which points to the buffer where the address is to be placed. On return, **len** specifies the number of bytes in the bound address and **buf** points to the bound address. If **maxlen** is not large enough to hold the returned address, an error results.

If the requested address is not available, or if no address is specified in **req** (the **len** field of **addr** in **req** is zero) the transport provider assigns an appropriate address to be bound, and returns that address in the **addr** field of **ret**. The user can compare the addresses in **req** and **ret** to determine whether the transport provider bound the transport endpoint to a different address than that requested.

req may be NULL if the user does not wish to specify an address to be bound. Here, the value of **qlen** is assumed to be zero, and the transport provider must assign an address to the transport endpoint. Similarly, **ret** may be NULL if the user does not care what address was bound by the provider and is not interested in the negotiated value of **qlen**. It is valid to set **req** and **ret** to NULL for the same call, in which case the provider chooses the address to bind to the transport endpoint and does not return that information to the user.

The **qlen** field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connect indications the transport provider should support for the given transport endpoint. An outstanding connect indication is one that has been passed to the transport user by the transport provider. A value of **qlen** greater than zero is only meaningful when issued by a passive transport user that expects other users to call it. The value of **qlen** is negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connect indications. On return, the **qlen** field in **ret** contains the negotiated value.

This function allows more than one transport endpoint to be bound to the same protocol address (however, the transport provider must support this capability also), but it is not allowable to bind more than one protocol address to the same transport endpoint. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connect indications associated with that protocol address. In other words, only one **t_bind** for a given protocol address may specify a value of **qlen** greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connect indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of **qlen** greater than zero, the transport provider assigns another address to be bound to that endpoint. If a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address is found to be busy for the duration of that connection. No other transport endpoints may be bound for listening while that initial listening endpoint is in the data transfer phase. This prevents more than one transport endpoint bound to the same protocol address from accepting connect indications.

On failure, **t_errno** may be set to one of the following:

[TACCES]	The user does not have permission to use the specified address.
[TBADADDR]	The specified protocol address was in an incorrect format or contained illegal information.
[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBUFOVFLW]	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The provider's state changes to T_IDLE and the information to be returned in **ret** is discarded.
[TNOADDR]	The transport provider could not allocate an address.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_open(3N), t_optmgmt(3N), t_unbind(3N)

DIAGNOSTICS

t_bind returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_close – close a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_close(fd)
int fd;

DESCRIPTION

The **t_close** function informs the transport provider that the user is finished with the transport endpoint specified by **fd**, and frees any local library resources associated with the endpoint. In addition, **t_close** closes the file associated with the transport endpoint.

t_close should be called from the **T_UNBND** state [see **t_getstate**(3N)]. However, this function does not check state information, so it may be called from any state to close a transport endpoint. If this occurs, the local library resources associated with the endpoint are freed automatically. In addition, **close**(2) is issued for that file descriptor; the close is abortive if no other process has that file open, and breaks any transport connection that may be associated with that endpoint.

On failure, **t_errno** may be set to the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

SEE ALSO

t_getstate(3N), t_open(3N), t_unbind(3N)

DIAGNOSTICS

t_close returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_connect – establish a connection with another transport user

SYNOPSIS

#include <tiuser.h>

int t_connect(fd, sndcall, rcvcall)
int fd;
struct t_call *sndcall;
struct t_call *rcvcall;

DESCRIPTION

This function enables a transport user to request a connection to the specified destination transport user. **fd** identifies the local transport endpoint where communication is established, while **sndcall** and **rcvcall** point to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

sndcall specifies information needed by the transport provider to establish a connection and **rcvcall** specifies information that is associated with the newly established connection.

Netbuf is described in **intro**(3). In **sndcall**, **addr** specifies the protocol address of the destination transport user, **opt** presents any protocol-specific information that might be needed by the transport provider, **udata** points to optional user data that may be passed to the destination transport user during connection establishment, and **sequence** has no meaning for this function.

On return in **rcvcall**, **addr** returns the protocol address associated with the responding transport endpoint, **opt** presents any protocol-specific information associated with the connection, **udata** points to optional user data that may be returned by the destination transport user during connection establishment, and **sequence** has no meaning for this function.

The **opt** argument implies no structure on the options that may be passed to the transport provider. The transport provider is free to specify the structure of any options passed to it. These options are specific to the underlying protocol of the transport provider. The user may choose not to negotiate protocol options by setting the **len** field of **opt** to zero. In this case, the provider may use default options.

The **udata** argument enables the caller to pass user data to the destination transport user and receive user data from the destination user during connection establishment. However, the amount of user data must not exceed the limits supported by the transport provider as returned by **t_open** (3N) or **t_getinfo** (3N). If the **len** [see **netbuf** in **intro**(3)] field of **udata** is zero in **sndcall**, no data is sent to the destination transport user.

On return, the **addr**, **opt**, and **udata** fields of **rcvcall** is updated to reflect values associated with the connection. Thus, the **maxlen** [see **netbuf** in **intro**(3)] field of each argument must be set before issuing this function to

indicate the maximum size of the buffer for each. However, **rcvcall** may be NULL, in which case no information is given to the user on return from **t_connect**.

By default, **t_connect** executes in synchronous mode, and waits for the destination user's response before returning control to the local user. A successful return (i.e. return value of zero) indicates that the requested connection has been established. However, if **O_NDELAY** is set (via **t_open** or **fcntl**), **t_connect** executes in asynchronous mode. In this case, the call will not wait for the remote user's response, but returns control immediately to the local user and return -1 with **t_errno** set to **TNODATA** to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connect request to the destination transport user.

On failure, **t_errno** may be set to one of the following:

[TACCES]	The user does not have permission to use the specified address or options.
[TBADADDR]	The specified protocol address was in an incorrect format or contained illegal information.
[TBADDATA]	The amount of user data specified was not within the bounds allowed by the transport provider.
[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBADOPT]	The specified protocol options were in an incorrect format or contained illegal information.
[TBUFOVFLW]	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. If executed in synchronous mode, the provider's state, as seen by the user, changes to **T_DATAXFER**, and the connect indication information to be returned in **rcvcall** is discarded.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNODATA]	**O_NDELAY** was set, so the function successfully initiated the connection establishment procedure, but did not wait for a response from the remote user.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_accept(3N), t_getinfo(3N), t_listen(3N), t_open(3N), t_optmgmt(3N), t_rcvconnect(3N)

DIAGNOSTICS

t_connect returns 0 on success and -1 on failure and **t_errno** is set.

NAME

t_error – produce error message

SYNOPSIS

#include <tiuser.h>

void t_error(errmsg)
char *errmsg;
extern int t_errno;
extern char *t_errlist[];
extern int t_nerr;

DESCRIPTION

t_error produces a message on the standard error output which describes the last error encountered during a call to a transport function. The argument string **errmsg** is a user-supplied error message that gives context to the error.

t_error prints the user-supplied error message followed by a colon, and the standard transport function error message for the current value contained in **t_errno**. If **t_errno** is **TSYSERR**, **t_error** also prints the standard error message for the current value contained in **errno** [see **intro**(2)].

t_errlist is the array of message strings, to allow user-message formatting. **t_errno** can be used as an index into this array to retrieve the error message string (without a terminating newline). **t_nerr** is the maximum index value for the **t_errlist** array.

t_errno is set when an error occurs and is not cleared on subsequent successful calls.

EXAMPLES

If a **t_connect** function fails on transport endpoint **fd2** because a bad address was given, the following call might follow the failure:

```
t_error("t_connect failed on fd2");
```

The diagnostic message would print as:

```
t_connect failed on fd2:  Incorrect transport address format
```

where `t_connect failed on fd2` tells the user which function failed on which transport endpoint, and `Incorrect transport address format` identifies the specific error that occurred.

NAME

t_free – free a library structure

SYNOPSIS

```
#include <tiuser.h>

int t_free(ptr, struct_type)
char *ptr;
int struct_type;
```

DESCRIPTION

The **t_free** function frees memory previously allocated by **t_alloc**. This function will free memory for the specified structure, and will also free memory for buffers referenced by the structure.

ptr points to one of the six structure types described for **t_alloc**, and **struct_type** identifies the type of that structure which can be one of the following:

T_BIND	**struct t_bind**
T_CALL	**struct t_call**
T_OPTMGMT	**struct t_optmgmt**
T_DIS	**struct t_discon**
T_UNITDATA	**struct t_unitdata**
T_UDERROR	**struct t_uderr**
T_INFO	**struct t_info**

where each of these structures is used as an argument to one or more transport functions.

t_free will check the **addr**, **opt**, and **udata** fields of the given structure (as appropriate), and free the buffers pointed to by the **buf** field of the **netbuf** [see **intro**(3)] structure. If **buf** is NULL, **t_free** will not attempt to free memory. After all buffers are freed, **t_free** will free the memory associated with the structure pointed to by **ptr**.

Undefined results will occur if **ptr** or any of the **buf** pointers points to a block of memory that was not previously allocated by **t_alloc**.

On failure, **t_errno** may be set to the following:

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_alloc(3N)

DIAGNOSTICS

t_free returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_getinfo – get protocol-specific service information

SYNOPSIS

```
#include <tiuser.h>

int t_getinfo(fd, info)
int fd;
struct t_info *info;
```

DESCRIPTION

This function returns the current characteristics of the underlying The **info** structure is used to return the same information returned by **t_open**. This function enables a transport user to access this information during any phase of communication.

This argument points to a **t_info** structure which contains the following members:

```
long addr;     /* max size of transport protocol address */
long options;  /* max number of bytes of protocol-specific options */
long tsdu;     /* max size transport service data unit (TSDU) */
long etsdu;    /* max size expedited transport service data unit */
long connect;  /* max data allowed on connection estab functions */
long discon;   /* max data allowed on t_snddis and t_rcvdis functions *
long servtype;/* service type supported by the transport provider */
```

The values of the fields have the following meanings:

addr A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of -1 specifies that there is no limit on the address size; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.

options A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.

tsdu A value greater than zero specifies the maximum size of a transport service data unit (**TSDU**); a value of zero specifies that the transport provider does not support the concept of **TSDU**, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of a **TSDU**; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu A value greater than zero specifies the maximum size of an expedited transport service data unit (**ETSDU**); a value of zero specifies that the transport provider does not support the concept of **ETSDU**, although it does support the sending of an expedited data stream with no logical boundaries preserved

across a connection; a value of -1 specifies that there is no limit on the size of an **ETSDU**; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

connect A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of -1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A value greater than or equal to zero specifies the maximum amount of data that may be associated with the **t_snddis** and **t_rcvdis** functions; a value of -1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of -2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

servtype This field specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the **t_alloc** function may be used to allocate these buffers. An error results if a transport user exceeds the allowed data size on any function. The value of each field may change as a result of option negotiation, and **t_getinfo** enables a user to retrieve the current characteristics.

DIAGNOSTICS

If successful, **t_getinfo** returns a 0. Otherwise –1 is returned, and **errno** is set to indicate the error.

The **servtype** field of **info** may specify one of the following values on return:

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS The transport provider supports a connectionless-mode service. For this service type, **t_open** returns -2 for **etsdu**, **connect**, and **discon**.

ERRORS

On failure, **t_errno** may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

t_open(3N)

NAME

t_getstate – get the current state

SYNOPSIS

#include <tiuser.h>

int t_getstate(fd)
int fd;

DESCRIPTION

The **t_getstate** function returns the current state of the provider associated with the transport endpoint specified by **fd.**

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TSTATECHNG]	The transport provider is undergoing a state change.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

t_open(3N)

DIAGNOSTICS

t_getstate returns the current state on successful completion and -1 on failure and **t_errno** is set to indicate the error. The current state may be one of the following:

T_UNBND	unbound
T_IDLE	idle
T_OUTCON	outgoing connection pending
T_INCON	incoming connection pending
T_DATAXFER	data transfer
T_OUTREL	outgoing orderly release (waiting for an orderly release indication)
T_INREL	incoming orderly release (waiting for an orderly release request)

If the provider is undergoing a state transition when **t_getstate** is called, the function fails.

NAME

t_listen – listen for a connect request

SYNOPSIS

```
#include <tiuser.h>

int t_listen(fd, call)
int fd;
struct t_call *call;
```

DESCRIPTION

This function listens for a connect request from a calling transport user. **fd** identifies the local transport endpoint where connect indications arrive, and on return, **call** contains information describing the connect indication. **call** points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

netbuf is described in **intro**(3). In **call**, **addr** returns the protocol address of the calling transport user, **opt** returns protocol-specific parameters associated with the connect request, **udata** returns any user data sent by the caller on the connect request, and **sequence** is a number that uniquely identifies the returned connect indication. The value of **sequence** enables the user to listen for multiple connect indications before responding to any of them.

Since this function returns values for the **addr**, **opt**, and **udata** fields of **call**, the **maxlen** [see **netbuf** in **intro**(3)] field of each must be set before issuing the **t_listen** to indicate the maximum size of the buffer for each.

By default, **t_listen** executes in synchronous mode and waits for a connect indication to arrive before returning to the user. However, if **O_NDELAY** is set (via **t_open** or **fcntl**), **t_listen** executes asynchronously, reducing to a poll for existing connect indications. If none are available, it returns -1 and sets **t_errno** to **TNODATA**.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBUFOVFLW]	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to T_INCON, and the connect indication information to be returned in **call** is discarded.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNODATA]	O_NDELAY was set, but no connect indications had been queued.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

WARNINGS

If a user issues **t_listen** in synchronous mode on a transport endpoint that was not bound for listening (i.e. **qlen** was zero on **t_bind**), the call waits forever because no connect indications arrive on that endpoint.

SEE ALSO

intro(3), t_accept(3N), t_bind(3N), t_connect(3N), t_open(3N), t_rcvconnect(3N)

DIAGNOSTICS

t_listen returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_look – look at the current event on a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_look(fd)
int fd;

DESCRIPTION

This function returns the current event on the transport endpoint specified by **fd**. This function enables a transport provider to notify a transport user of an asynchronous event when the user is issuing functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, **TLOOK**, on the current or next function to be executed.

This function also enables a transport user to poll a transport endpoint periodically for asynchronous events.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

t_open(3N)

DIAGNOSTICS

Upon success, **t_look** returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists. One of the following events is returned:

T_CONNECT	connect confirmation received
T_DATA	normal data received
T_DISCONNECT	disconnect received
T_ERROR	fatal error indication
T_EXDATA	expedited data received
T_LISTEN	connection indication received
T_ORDREL	orderly release indication
T_UDERR	datagram error indication

On failure, -1 is returned and **t_errno** is set to indicate the error.

NAME

t_open – establish a transport endpoint

SYNOPSIS

```
#include <tiuser.h>

int t_open(path, oflag, info)
char *path;
int oflag;
struct t_info *info;
```

DESCRIPTION

t_open must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by opening a file that identifies a particular transport provider (i.e. transport protocol) and returning a file descriptor that identifies that endpoint.

path points to the pathname of the file to open, and **oflag** identifies any open flags [as in **open**(2)]. **t_open** returns a file descriptor that is used by all subsequent functions to identify the particular local transport endpoint.

This function also returns various default characteristics of the underlying transport protocol by setting fields in the **t_info** structure. This argument points to a **t_info** which contains the following members:

```
long addr;     /* max size of the transport protocol address */
long options;  /* max number of bytes of protocol-specific options */
long tsdu;     /* max size of a transport service data unit (TSDU) */
long etsdu;    /* max size of an expedited transport service data
                  unit (ETSDU) */
long connect;  /* max amount of data allowed on connection
                  establishment functions */
long discon;   /* max amount of data allowed on t_snddis and
                  t_rcvdis functions */
long servtype; /* service type supported by the transport provider */
```

The values of the fields have the following meanings:

addr A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of -1 specifies that there is no limit on the address size; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.

options A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.

tsdu A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a

value of -1 specifies that there is no limit on the size of a **TSDU**; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu A value greater than zero specifies the maximum size of an expedited transport service data unit (**ETSDU**); a value of zero specifies that the transport provider does not support the concept of **ETSDU**, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of an **ETSDU**; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

connect A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of -1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A value greater than or equal to zero specifies the maximum amount of data that may be associated with the **t_snddis** and **t_rcvdis** functions; a value of -1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of -2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

servtype This field specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the **t_alloc** function may be used to allocate these buffers. An error results if a transport user exceeds the allowed data size on any function.

The **servtype** field of **info** may specify one of the following values on return:

T_CLTS The transport provider supports a connectionless-mode service. For this service type, **t_open** returns -2 for **etsdu**, **connect**, and **discon**.

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports a connection-mode service with the optional orderly release facility.

A single transport endpoint may support only one of the above services at one time.

If **info** is set to **NULL** by the transport user, no protocol information is returned by **t_open**.

On failure, **t_errno** may be set to the following:

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

open(2)

DIAGNOSTICS

t_open returns a valid file descriptor on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_optmgmt – manage options for a transport endpoint

SYNOPSIS

```
#include <tiuser.h>

int t_optmgmt(fd, req, ret)
int fd;
struct t_optmgmt *req;
struct t_optmgmt *ret;
```

DESCRIPTION

The **t_optmgmt** function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. **fd** identifies a bound transport endpoint.

The **req** and **ret** arguments point to a **t_optmgmt** structure containing the following members:

```
struct netbuf opt;
long  flags;
```

The **opt** field identifies protocol options and the **flags** field is used to specify the action to take with those options.

The options are represented by a **netbuf** [see **intro**(3); also for **len, buf** and **maxlen**] structure in a manner similar to the address in **t_bind**. **req** is used to request a specific action of the provider and to send options to the provider. **len** specifies the number of bytes in the options, **buf** points to the options buffer, and **maxlen** has no meaning for the **req** argument. The transport provider may return options and flag values to the user through **ret**. For **ret**, **maxlen** specifies the maximum size of the options buffer and **buf** points to the buffer where the options are to be placed. On return, **len** specifies the number of bytes of options returned. **maxlen** has no meaning for the **req** argument, but must be set in the **ret** argument to specify the maximum number of bytes the options buffer can hold. The actual structure and content of the options is imposed by the transport provider.

The **flags** field of **req** can specify one of the following actions:

T_CHECK — This action enables the user to verify whether the options specified in **req** are supported by the transport provider. On return, the **flags** field of **ret** has either T_SUCCESS or T_FAILURE set to indicate to the user whether the options are supported. These flags are only meaningful for the T_CHECK request.

T_DEFAULT — This action enables a user to retrieve the default options supported by the transport provider into the **opt** field of **ret**. In **req**, the **len** field of **opt** must be zero and the **buf** field may be NULL.

T_NEGOTIATE — This action enables the user to negotiate the values of the options specified in **req** with the transport provider. The provider evaluates the requested options and negotiates the values, returning the negotiated values through **ret**.

If issued as part of the connectionless-mode service, **t_optmgmt** may block due to flow control constraints. The function is not complete until the transport provider has processed all previously sent data units.

On failure, **t_errno** may be set to one of the following:

[TACCES]	The user does not have permission to negotiate the specified options.
[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBADFLAG]	An invalid flag was specified.
[TBADOPT]	The specified protocol options were in an incorrect format or contained illegal information.
[TBUFOVFLW]	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in **ret** is discarded.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_getinfo(3N), t_open(3N)

DIAGNOSTICS

t_optmgmt returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_rcv – receive data or expedited data sent over a connection

SYNOPSIS

```
int t_rcv(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int *flags;
```

DESCRIPTION

This function receives either normal or expedited data. **fd** identifies the local transport endpoint through which data will arrive, **buf** points to a receive buffer where user data will be placed, and **nbytes** specifies the size of the receive buffer. **flags** may be set on return from **t_rcv** and specifies optional flags as described below.

By default, **t_rcv** operates in synchronous mode and waits for data to arrive if none is currently available. However, if O_NDELAY is set (via **t_open** or **fcntl**), **t_rcv** executes in asynchronous mode and fails if no data is available. (See TNODATA below.)

On return from the call, if T_MORE is set in **flags** this indicates that there is more data and the current transport service data unit (*TSDU*) or expedited transport service data unit (*ETSDU*) must be received in multiple **t_rcv** calls. Each **t_rcv** with the T_MORE flag set indicates that another **t_rcv** must follow immediately to get more data for the current **TSDU** The end of the **TSDU** is identified by the return of a **t_rcv** call with the T_MORE flag not set. If the transport provider does not support the concept of a **TSDU** as indicated in the **info** argument on return from **t_open** or **t_getinfo**, the T_MORE flag is not meaningful and should be ignored.

On return, the data returned is expedited data if T_EXPEDITED is set in **flags**. If the number of bytes of expedited data exceeds **nbytes**, **t_rcv** is set T_EXPEDITED and T_MORE on return from the initial call. Subsequent calls to retrieve the remaining **ETSDU** will not have T_EXPEDITED set on return. The end of the ETSDU is identified by the return of a **t_rcv** call with the T_MORE flag not set.

If expedited data arrives after part of a TSDU has been retrieved, receipt of the remainder of the TSDU is suspended until the **ETSDU** has been processed. Only after the full **ETSDU** has been retrieved (T_MORE not set) is the remainder of the **TSDU** be available to the user.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNODATA]	O_NDELAY was set, but no data is currently available from the transport provider.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

t_open(3N), t_snd(3N)

DIAGNOSTICS

On successful completion, **t_rcv** returns the number of bytes received; on failure it returns -1 and **t_errno** is set to indicate the error.

NAME

t_rcvconnect – receive the confirmation from a connect request

SYNOPSIS

#include <tiuser.h>

int t_rcvconnect(fd, call)
int fd;
struct t_call *call;

DESCRIPTION

This function enables a calling transport user to determine the status of a previously sent connect request and is used in conjunction with **t_connect** to establish a connection in asynchronous mode. The connection is established on successful completion of this function.

fd identifies the local transport endpoint where communication will be established, and **call** contains information associated with the newly established connection. **call** points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

netbuf is described in **intro**(3). In **call, addr** returns the protocol address associated with the responding transport endpoint, **opt** presents any protocol-specific information associated with the connection, **udata** points to optional user data that may be returned by the destination transport user during connection establishment, and **sequence** has no meaning for this function.

The **maxlen** [see **netbuf** in **intro**(3)] field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, **call** may be NULL, in which case no information is given to the user on return from **t_rcvconnect**. By default, **t_rcvconnect** executes in synchronous mode and waits for the connection to be established before returning. On return, the **addr, opt**, and **udata** fields reflect values associated with the connection.

If O_NDELAY is set (via **t_open** or **fcntl**), **t_rcvconnect** executes in asynchronous mode, and reduces to a poll for existing connect confirmations. If none are available, **t_rcvconnect** fails and returns immediately without waiting for the connection to be established. (See TNODATA below.) **t_rcvconnect** must be re-issued at a later time to complete the connection establishment phase and retrieve the information returned in **call**.

On failure, **t_errno** may be set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TBUFOVFLW] The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument and the connect information to be returned in **call** is discarded. The provider's state, as seen by the user, is changed to DATAXFER.

[TLOOK]	An asynchronous event has occurred on this transport connection and requires immediate attention.
[TNODATA]	O_NDELAY was set, but a connect confirmation has not yet arrived.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_accept(3N), t_bind(3N), t_connect(3N), t_listen(3N), t_open(3N)

DIAGNOSTICS

t_rcvconnect returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_rcvdis – retrieve information from disconnect

SYNOPSIS

#include <tiuser.h>

t_rcvdis(fd, discon)
int fd;
struct t_discon *discon;

DESCRIPTION

This function is used to identify the cause of a disconnect, and to retrieve any user data sent with the disconnect. **fd** identifies the local transport endpoint where the connection existed, and **discon** points to a **t_discon** structure containing the following members:

```
struct netbuf udata;
int reason;
int sequence;
```

netbuf is described in **intro**(3). **reason** specifies the reason for the disconnect through a protocol-dependent reason code, **udata** identifies any user data that was sent with the disconnect, and **sequence** may identify an outstanding connect indication with which the disconnect is associated. **sequence** is only meaningful when **t_rcvdis** is issued by a passive transport user who has executed one or more **t_listen** functions and is processing the resulting connect indications. If a disconnect indication occurs, **sequence** can be used to identify which of the outstanding connect indications is associated with the disconnect.

If a user does not care if there is incoming data and does not need to know the value of **reason** or **sequence**, **discon** may be NULL and any user data associated with the disconnect will be discarded. However, if a user has retrieved more than one outstanding connect indication (via **t_listen**) and **discon** is NULL, the user will be unable to identify with which connect indication the disconnect is associated.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TNODIS]	No disconnect indication currently exists on the specified transport endpoint.
[TBUFOVFLW]	The number of bytes allocated for incoming data is not sufficient to store the data. The provider's state, as seen by the user, will change to T_IDLE, and the disconnect indication information to be returned in **discon** will be discarded.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_connect(3N), t_listen(3N), t_open(3N), t_snddis(3N)

DIAGNOSTICS

t_rcvdis returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_rcvrel – acknowledge receipt of an orderly release indication

SYNOPSIS

#include <tiuser.h>

t_rcvrel(fd)
int fd;

DESCRIPTION

This function is used to acknowledge receipt of an orderly release indication. **fd** identifies the local transport endpoint where the connection exists. After receipt of this indication, the user may not attempt to receive more data because such an attempt blocks forever. However, the user may continue to send data over the connection if **t_sndrel** has not been issued by the user.

This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on **t_open** or **t_getinfo**.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNOREL]	No orderly release indication currently exists on the specified transport endpoint.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

t_open(3N), t_sndrel(3N)

DIAGNOSTICS

t_rcvrel returns 0 on success, and -1 on failure **t_errno** is set to indicate the error.

NAME

t_rcvudata – receive a data unit

SYNOPSIS

```
#include <tiuser.h>

int t_rcvudata(fd, unitdata, flags)
int fd;
struct t_unitdata *unitdata;
int *flags;
```

DESCRIPTION

This function is used in connectionless mode to receive a data unit from another transport user. **fd** identifies the local transport endpoint through which data will be received, **unitdata** holds information associated with the received data unit, and **flags** is set on return to indicate that the complete data unit was not received. **unitdata** points to a **t_unitdata** structure containing the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
```

The **maxlen** [see **netbuf** in **intro**(3)] field of **addr**, **opt**, and **udata** must be set before issuing this function to indicate the maximum size of the buffer for each.

On return from this call, **addr** specifies the protocol address of the sending user, **opt** identifies protocol-specific options that were associated with this data unit, and **udata** specifies the user data that was received.

By default, **t_rcvudata** operates in synchronous mode and waits for a data unit to arrive if none is currently available. However, if **O_NDELAY** is set (via **t_open** or **fcntl**), **t_rcvudata** executes in asynchronous mode and fails if no data units are available.

If the buffer defined in the **udata** field of **unitdata** is not large enough to hold the current data unit, the buffer is filled and T_MORE is set in **flags** on return to indicate that another **t_rcvudata** should be issued to retrieve the rest of the data unit. Subsequent **t_rcvudata** call(s) return zero for the length of the address and options until the full data unit has been received.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBUFOVFLW]	The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data information to be returned in **unitdata** is discarded.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNODATA]	O_NDELAY was set, but no data units are currently available from the transport provider.

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_rcvuderr(3N), t_sndudata(3N)

DIAGNOSTICS

t_rcvudata returns 0 on successful completion and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_rcvuderr – receive a unit data error indication

SYNOPSIS

```
#include <tiuser.h>

int t_rcvuderr(fd, uderr)
int fd;
struct t_uderr *uderr;
```

DESCRIPTION

This function is used in connectionless mode to receive information concerning an error on a previously sent data unit, and should only be issued following a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error. **fd** identifies the local transport endpoint through which the error report will be received, and **uderr** points to a **t_uderr** structure containing the following members:

```
struct netbuf addr;
struct netbuf opt;
long  error;
```

netbuf is described in **intro**(3). The **maxlen** [see **netbuf** in **intro**(3)] field of **addr** and **opt** must be set before issuing this function to indicate the maximum size of the buffer for each.

On return from this call, the **addr** structure specifies the destination protocol address of the erroneous data unit, the **opt** structure identifies protocol-specific options that were associated with the data unit, and **error** specifies a protocol-dependent error code.

If the user does not care to identify the data unit that produced an error, **uderr** may be set to NULL and **t_rcvuderr** simply clears the error indication without reporting any information to the user.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBUFOVFLW]	The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data error information to be returned in **uderr** is discarded.
[TNOUDERR]	No unit data error indication currently exists on the specified transport endpoint.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_rcvudata(3N), t_sndudata(3N)

DIAGNOSTICS

t_rcvuderr returns 0 on successful completion and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_snd – send data or expedited data over a connection

SYNOPSIS

```
#include <tiuser.h>

int t_snd(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int *flags;
```

DESCRIPTION

This function is used to send either normal or expedited data. **fd** identifies the local transport endpoint over which data should be sent, **buf** points to the user data, **nbytes** specifies the number of bytes of user data to be sent, and **flags** specifies any optional flags described below.

By default, **t_snd** operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NDELAY is set (via **t_open** or **fcntl**), **t_snd** executes in asynchronous mode, and fails immediately if there are flow control restrictions.

Even when there are no flow control restrictions, **t_snd** waits if STREAMS internal resources are not available, regardless of the state of O_NDELAY.

On successful completion, **t_snd** returns the number of bytes accepted by the transport provider. Normally this equals the number of bytes specified in **nbytes**. However, if O_NDELAY is set , it is possible that only part of the data will be accepted by the transport provider. In this case, **t_snd** sets **T_MORE** for the data that was sent (see below), and returns a value less than **nbytes**. If **nbytes** is zero, no data is passed to the provider and **t_snd** returns zero.

If **T_EXPEDITED** is set in **flags**, the data is sent as expedited data, and is subject to the interpretations of the transport provider.

If **T_MORE** is set in **flags**, or is set as described above, an indication is sent to the transport provider that the transport service data unit (TSDU) or expedited transport service data unit (ETSDU) is being sent through multiple **t_snd** calls. Each **t_snd** with the **T_MORE** flag set indicates that another **t_snd** follows with more data for the current TSDU. The end of the TSDU (or ETSDU) is identified by a **t_snd** call with the **T_MORE** flag not set. Use of **T_MORE** enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the **info** argument on return from **t_open** or **t_getinfo**, the **T_MORE** flag is not meaningful and should be ignored.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as returned by **t_open** or **t_getinfo**. If the size is exceeded, a TSYSERR with system error EPROTO occurs. However, the **t_snd** may not fail because EPROTO errors may not be reported immediately. In this case, a subsequent call that accesses the transport endpoint fails with the associated TSYSERR.

If **t_snd** is issued from the **T_IDLE** state, the provider may silently discard the data. If **t_snd** is issued from any state other than **T_DATAXFER, T_INREL,** or **T_IDLE,** the provider generates a TSYSERR with system error EPROTO (which may be reported in the manner described above).

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TFLOW]	O_NDELAY was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error [see **intro**(2)] has been detected during execution of this function.

SEE ALSO

t_open(3N), t_rcv(3N)

DIAGNOSTICS

On successful completion, **t_snd** returns the number of bytes accepted by the transport provider; on failure it returns –1 and **t_errno** is set to indicate the error.

NAME

t_snddis – send user-initiated disconnect request

SYNOPSIS

```
#include <tiuser.h>

int t_snddis(fd, call)
int fd;
struct t_call *call;
```

DESCRIPTION

This function is used to initiate an abortive release on an already established connection or to reject a connect request. **fd** identifies the local transport endpoint of the connection, and **call** specifies information associated with the abortive release. **call** points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

netbuf is described in **intro**(3). The values in **call** have different semantics, depending on the context of the call to **t_snddis**. When rejecting a connect request, **call** must be non-NULL and contain a valid value of **sequence** to uniquely identify the rejected connect indication to the transport provider. The **addr** and **opt** fields of **call** are ignored. In all other cases, **call** need only be used when data is being sent with the disconnect request. The **addr, opt,** and **sequence** fields of the **t_call** structure are ignored. If the user does not wish to send data to the remote user, the value of **call** may be NULL.

udata specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider as returned by **t_open** or **t_getinfo**. If the **len** field of **udata** is zero, no data is sent to the remote user.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBADDATA]	The amount of user data specified was not within the bounds allowed by the transport provider. The transport provider's outgoing queue is flushed, so data may be lost.
[TBADSEQ]	An invalid sequence number was specified, or a NULL call structure was specified when rejecting a connect request. The transport provider's outgoing queue is flushed, so data may be lost.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TOUTSTATE]	The function was issued in the wrong sequence. The transport provider's outgoing queue may be flushed, so

data may be lost.

[TSYSERR] A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_connect(3N), t_getinfo(3N), t_listen(3N), t_open(3N)

DIAGNOSTICS

t_snddis returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_sndrel – initiate an orderly release

SYNOPSIS

#include <tiuser.h>

int t_sndrel(fd)
int fd;

DESCRIPTION

This function is used to initiate an orderly release of a transport connection and indicates to the transport provider that the transport user has no more data to send. **fd** identifies the local transport endpoint where the connection exists. After issuing **t_sndrel**, the user may not send any more data over the connection. However, a user may continue to receive data if an orderly release indication has been received.

This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on **t_open** or **t_getinfo**.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TFLOW]	O_NDELAY was set, but the flow control mechanism prevented the transport provider from accepting the function at this time.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

t_open(3N), t_rcvrel(3N)

DIAGNOSTICS

t_sndrel returns 0 on success and -1 on failure and **t_errno** is set to indicate the error.

NAME

t_sndudata – send a data unit

SYNOPSIS

#include <tiuser.h>

int t_sndudata(fd, unitdata)
int fd;
struct t_unitdata *unitdata;

DESCRIPTION

This function is used in connectionless mode to send a data unit to another transport user. **fd** identifies the local transport endpoint through which data will be sent, and **unitdata** points to a **t_unitdata** structure containing the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
```

netbuf is described in **intro**(3). In **unitdata**, **addr** specifies the protocol address of the destination user, **opt** identifies protocol-specific options that the user wants associated with this request, and **udata** specifies the user data to be sent. The user may choose not to specify what protocol options are associated with the transfer by setting the **len** field of **opt** to zero. In this case, the provider may use default options.

If the **len** field of **udata** is zero, no data unit is passed to the transport provider; **t_sndudata** will not send zero-length data units.

By default, **t_sndudata** operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NDELAY is set (via **t_open** or **fcntl**), **t_sndudata** executes in asynchronous mode and fails under such conditions.

If **t_sndudata** is issued from an invalid state, or if the amount of data specified in **udata** exceeds the TSDU size as returned by **t_open** or **t_getinfo**, the provider generates an EPROTO protocol error. (See TSYSERR below.)

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TFLOW]	O_NDELAY was set, but the flow control mechanism prevented the transport provider from accepting data at this time.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

intro(3), t_rcvudata(3N), t_rcvuderr(3N)

DIAGNOSTICS

t_sndudata returns 0 on successful completion and -1 on failure **t_errno** is set to indicate the error.

NAME

t_sync – synchronize transport library

SYNOPSIS

```
#include <tiuser.h>

int t_sync(fd)
int fd;
```

DESCRIPTION

For the transport endpoint specified by **fd**, **t_sync** synchronizes the data structures managed by the transport library with information from the underlying transport provider. In doing so, it can convert a raw file descriptor [obtained via **open**(2), **dup**(2), or as a result of a **fork**(2) and **exec**(2)] to an initialized transport endpoint, assuming that file descriptor referenced a transport provider. This function also allows two cooperating processes to synchronize their interaction with a transport provider.

For example, if a process **forks** a new process and issues an **exec**, the new process must issue a **t_sync** to build the private library data structure associated with a transport endpoint and to synchronize the data structure with the relevant provider information.

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. If multiple processes are using the same endpoint, they should coordinate their activities so as not to violate the state of the provider. **t_sync** returns the current state of the provider to the user, thereby enabling the user to verify the state before taking further action. This coordination is only valid among cooperating processes; it is possible that a process or an incoming event could change the provider's state **after** a **t_sync** is issued.

If the provider is undergoing a state transition when **t_sync** is called, the function fails.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor is a valid open file descriptor but does not refer to a transport endpoint.
[TSTATECHNG]	The transport provider is undergoing a state change.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

dup(2), exec(2), fork(2), open(2)

DIAGNOSTICS

t_sync returns the state of the transport provider on successful completion and -1 on failure and **t_errno** is set to indicate the error. The state returned may be one of the following:

T_DATAXFER	data transfer
T_IDLE	idle
T_INCON	incoming connection pending

T_INREL	incoming orderly release (waiting for an orderly release request)
T_OUTCON	outgoing connection pending
T_OUTREL	outgoing orderly release (waiting for an orderly release indication)
T_UNBND	unbound

NAME

t_unbind – disable a transport endpoint

SYNOPSIS

#include <tiuser.h>

int t_unbind(fd)
int fd;

DESCRIPTION

The **t_unbind** function disables the transport endpoint specified by **fd** which was previously bound by **t_bind** (3N). On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider.

On failure, **t_errno** may be set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TLOOK]	An asynchronous event has occurred on this transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TSYSERR]	A system error has occurred during execution of this function.

SEE ALSO

t_bind(3N)

DIAGNOSTICS

t_unbind returns 0 on success and -1 on failure; **t_errno** is set to indicate the error.

NAME

intro – introduction to Parallel Programming Library

DESCRIPTION

These routines constitute the Parallel Programming Library, which supports microtasking and multitasking in C, Pascal, and FORTRAN programs. (For information on microtasking and multitasking programming models, refer to the *Guide to Parallel Programming*.)

The routines are contained in the file */usr/lib/libpps.a*. These routines are linked into a program by including the **-lpps** and **-lseq** options in the **cc** or **ld** command line, or by including the **-lpps** and **-lseq** options, or the **-mp** option in the **fortran** command line.

For an overview of how the Parallel Programming Library routines are used, and for sample programs and related information, refer to the *Guide to Parallel Programming*.

LIST OF FUNCTIONS

The following routines support microtasking:

Name	*Appears on Page*	*Description*
m_fork	m_fork.3pps	execute a subprogram in parallel
m_get_myid	m_get_myid.3pps	return process identification
m_get_numprocs	m_get_numprocs.3pps	get number of child processes
m_kill_procs	m_kill_procs.3pps	kill child processes
m_lock	m_lock.3pps	initialize and lock a lock
m_multi	m_single.3pps	end single-process section
m_next	m_next.3pps	increment global counter
m_park_procs	m_park_procs.3pps	suspend child process execution
m_rele_procs	m_park_procs.3pps	resume child process execution
m_set_procs	m_set_procs.3pps	set number of child processes
m_single	m_single.3pps	start single-process section
m_sync	m_sync.3pps	check in at barrier
m_unlock	m_lock.3pps	unlock a lock

The following routines support multitasking:

Name	*Appears on Page*	*Description*
cpus_online	cpus_online.3pps	return number of CPUs on-line
s_clock	s_lock.3pps	lock a lock, return if unsuccessful
S_CLOCK	s_lock.3pps	lock a lock, return if unsuccessful (C macro)
s_init_barrier	s_wait_barrier.3pps	initialize a barrier
S_INIT_BARRIER	s_wait_barrier.3pps	initialize a barrier (C macro)
s_init_lock	s_lock.3pps	initialize a lock
S_INIT_LOCK	s_lock.3pps	initialize a lock (C macro)
s_lock	s_lock.3pps	lock a lock
S_LOCK	s_lock.3pps	lock a lock (C macro)
s_unlock	s_lock.3pps	unlock a lock
S_UNLOCK	s_lock.3pps	unlock a lock (C macro)
s_wait_barrier	s_wait_barrier.3pps	wait at a barrier

S_WAIT_BARRIER	s_wait_barrier.3pps	wait at a barrier (C macro)

The following routines support memory allocation for parallel programming. The **brk** and **sbrk** routines are available without loading the Parallel Programming library (see **brk**(2)), but the versions in the Parallel Programming library are necessary for compatibility with the rest of the library.

Name	*Appears on Page*	*Description*
brk	brk.3pps	change private data segment size
sbrk	brk.3pps	change private data segment size
shbrk	shbrk.3pps	change shared data segment size
shfree	shmalloc.3pps	deallocate shared data memory
shmalloc	shmalloc.3pps	allocate shared data memory
shrealloc	shmalloc.3pps	reallocate shared data memory
shsbrk	shbrk.3pps	change shared data segment size

NAME

brk, sbrk – change private data segment size

SYNOPSIS

C syntax:

```
#include <parallel/parallel.h>
caddr_t brk(addr)
caddr_t addr;

caddr_t sbrk(incr)
int incr;
```

FORTRAN syntax

none

DESCRIPTION

These routines are identical to the standard **brk** and **sbrk** (see **brk(2)**) routines except that these routines verify that the private data segment does not overlap the shared data segment which follows it. If the **brk** or **sbrk** call will cause the segments to overlap, the routine issues an error.

ERRORS

If an error occurs, the return value is -1 and the variable **errno** contains the error code. **brk** and **sbrk** can return the following error codes:

[ENOMEM] The requested private data segment will overlap the shared data segment.

[ENOMEM] The routine cannot allocate file system space to expand the file which has been memory mapped to the shared data segment (see **mmap(2SEQ)**). The routine tries to allocate file system space from directories in the following order:

1. directory indicated by **TMPPATH** environment variable
2. current directory
3. user's home directory
4. directory */usr/tmp*
5. directory */tmp*

SEE ALSO

execve(2), malloc(3), end(3), vm_ctl(2SEQ)

NOTES

The gap between the private data segment and the shared data segment can be adjusted by modifying the file */usr/sequent/gtpp.demo/ifile*, and then including the ifile on the compiler command line as follows:

```
cc –o foo foo.c –lpps –lseq ifile
```

Refer to the linker documentation in the *Language Tools* binder for more information about ifiles.

When **brk** and **sbrk** increase a program's address space, they also attempt to increase its allowable resident set size (see **vm_ctl(2SEQ)**).

NAME

cpus_online – returns the number of CPUs on-line

SYNOPSIS

C syntax

int cpus_online ();

FORTRAN syntax

integer*4 function cpus_online

DESCRIPTION

The **cpus_online** routine returns the number of processors currently configured and on-line.

SEE ALSO

tmp_ctl(2seq)

NAME

m_fork – execute a subprogram in parallel

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
m_fork(func[,arg,...]);
void (*func)();
sometype args;
```

FORTRAN syntax

```
external func
integer*4 m_fork
i=m_fork(func[,arg,...])
subroutine func(arg,...)
```

DESCRIPTION

The **m_fork** routine assigns a subprogram to child processes, which then cooperate in executing the subprogram in parallel. The number of child processes used by the **m_fork** call can be set with a previous call to **m_set_procs**. If **m_set_procs** has not been called, the number of child processes defaults to *(number of CPUs on-line) /2*. If the program has no child processes from previous **m_fork** calls, the call creates the child processes. If there are already child processes from a previous call, **m_fork** re-uses the existing processes.

When an **m_fork** call creates child processes, each child process is given a private integer variable called *m_myid*, which identifies it within the set of child processes being created. The parent process's identification number is always zero. The first child process's identification is 1, the second's is 2, and so on. You can call the routine **m_getmyid** to find out the identification number of a child process (see **m_getmyid**(3PPS)).

For C programs, the header file */usr/include/parallel/microtask.h* contains an external declaration of the variable *m_myid* and the variable *m_numprocs*, which indicates the total number of processes executing the subprogram (including all the child processes and the parent process).

Once child processes are available, **m_fork** starts them executing the subprogram **func** with the given arguments. (For Pascal programs, **func** must be an inner scope procedure.) The child processes execute the subprogram until they all return from it. At this point, the program returns from the **m_fork** call and the child processes spin, waiting for more work. The program can either kill the child processes with a call to the routine **m_kill**, suspend them with a call to **m_park_procs**, or let the child processes spin until they are re-used by another **m_fork** call. If the child processes are to be re-used, the **m_park_procs** offers the most efficient use of the Sequent system, because it saves the CPU usage of having the processes spin and it saves the overhead of having to recreate processes on the next **m_fork** call.

You must ensure that arguments passed to the subprogram **func** are either call-by-value arguments or addresses of data in shared memory. They must not be addresses in the parent's private data segment.

ERRORS

The **m_fork** call fails and no child processes are created if any of the following

error conditions occurs:

[EAGAIN] The **m_fork** call would exceed *nproc*, the system's limit on the total number of executing processes.

[EAGAIN] The **m_fork** call would exceed *maxuprc*, the system's limit on executing processes for a single user.

[EINVAL] This call to **m_fork** is nested within a previous call.

NOTES

Each call to **m_fork** resets the global counter (see **m_next**(3PPS)).

SEE ALSO

shmalloc(3PPS), brk(3PPS), m_set_procs(3PPS), m_kill_procs(3PPS), m_next(3PPS)

NAME

m_get_myid – return process identification

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
int m_get_myid();
int i;
i=m_get_myid();
```

FORTRAN syntax

```
integer*4 function m_get_myid
```

DESCRIPTION

The **m_get_myid** routine returns the value of the variable *m_myid*, the process's indentification number. For the parent process, this variable has the value zero. Child processes are assigned identification numbers in the order of their creation: the first child process has identification number 1, the second has 2, and so on. The C header file */usr/include/parallel/microtask.h* contains an external declaration of the variable.

SEE ALSO

fortran(1), m_fork(3PPS)

NAME

m_get_numprocs – return number of child processes

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
int m_get_numprocs();
```

FORTRAN syntax

```
integer*4 function m_get_numprocs
```

DESCRIPTION

The **m_get_numprocs** routine returns the value of the variable *m_numprocs*, the current number of processes in the program. This value reflects the number of child processes plus one, the parent process.

SEE ALSO

fortran(1), m_set_procs(3PPS)

NAME

m_kill_procs – kill child processes

SYNOPSIS

C syntax:

#include <parallel/microtask.h>
m_kill_procs();

FORTRAN syntax

subroutine m_kill_procs

DESCRIPTION

The **m_kill_procs** routine terminates the child processes created by a previous call to **m_fork**.

The **m_kill_procs** routine fails if it is called from a subprogram invoked by an **m_fork** call.

ERRORS

The **m_kill_procs** call fails if the following error condition occurs:

[EINVAL] Some child processes are still executing within an **m_fork** call.

SEE ALSO

m_fork(3PPS)

NAME

m_lock, m_unlock – lock, unlock locks

SYNOPSIS

C syntax:

#include <parallel/microtask.h>
m_lock ();
m_unlock ();

FORTRAN syntax

subroutine m_lock()
subroutine m_unlock()

DESCRIPTION

The **m_lock** and **m_unlock** routines are microtasking interfaces to a single **slock_t**-type lock. For a single lock, they are easier to use than the **s_init_lock**, **s_lock**, and **s_unlock** routines because they don't require you to declare or initialize the lock before using it. They are also faster than the other routines because they do not pass the lock address as an argument.

m_lock locks the lock. **m_lock** is always successful; it spins as long as is necessary to acquire the lock.

m_unlock unlocks the lock.

SEE ALSO

intro(3PPS), s_lock(3PPS), shmalloc(3PPS)

NAME

m_next – increment global counter

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
int m_next();
```

FORTRAN syntax

```
integer*4 function m_next
```

DESCRIPTION

The **m_next** routine atomically increments a global counter. The program's first call to **m_next** returns the value 1, the second returns 2, and so on. Calls to the **m_fork**, **m_sync**, or **m_single** routines reset the global counter to zero.

SEE ALSO

m_fork(3PPS), m_sync(3PPS), m_single(3PPS)

NAME

m_park_procs, m_rele_procs – suspend and resume child process execution

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
m_park_procs();
(serial code)
m_rele_procs();
```

FORTRAN syntax

```
subroutine m_park_procs
subroutine m_rele_procs
```

DESCRIPTION

The **m_park_procs** routine suspends execution of child processes created by an **m_fork** call. Typically, you would suspend child processes while the parent process is doing extensive I/O or setting up another phase of the program. The **m_rele_procs** routine resumes child process execution when the child processes are again required.

Do not call **m_park_procs** when **m_fork** is executing. Likewise, do not call **m_park_procs** when the child processes are already suspended. To suspend child process execution within an **m_fork** call, use **m_single** and **m_next**.

ERRORS

These routines can return the following error:

[EINVAL] The routine was called in an inappropriate context. For example, the routine was called from a subprogram executing in an **m_fork** call, or the processes were already suspended.

SEE ALSO

m_single(3PPS)

NAME

m_set_procs – set number of child processes

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
int m_set_procs(nprocs);
int nprocs;
```

FORTRAN syntax

```
integer*4 function m_set_procs(nprocs)
integer*4 nprocs
```

DESCRIPTION

The **m_set_procs** routine declares the number of processes to execute subprograms in parallel on subsequent calls to **m_fork**. The argument **nprocs** declares the total number of processes that will run in parallel, including the parent process and the child processes. If **nprocs** is zero, the program creates no child processes, but all barriers and locks are initialized as if the program were going to create child processes.

The **m_set_procs** routine initializes a shared variable called **m_numprocs**, which controls the number of processes created by subsequent calls to **m_fork**. The C header file */usr/include/parallel/microtask.h* contains an external declaration of **m_numprocs**, as well as the constant MAXPROCS which determines the maximum number of processes that the system will allow the program to create. The other limiting factor is the number of CPUs on-line: **m_nprocs** can be no more than the number of CPUs on-line minus one.

The **m_set_procs** routine is optional: if the program does not call this routine before calling **m_fork**, the number of processes defaults to **(number of CPUs on-line)/2**. The program must not call **m_set_procs** while the child processes from an **m_fork** call are still alive (that is, before an **m_kill_procs** call to kill the child processes).

ERRORS

If an **m_set_procs** call is successful, the return value is zero. If the call fails, the return value is -1 and the variable **errno** holds the error code.

The **m_set_procs** call fails if one of the following error conditions occurs:

[EINVAL]	The argument **nprocs** is greater than MAXPROCS or it is greater than or equal to the number of on-line CPUs.
[EINVAL]	Children from an **m_fork** call are already running. In this case, call **m_kill_procs** to kill the existing child processes before calling **m_set_procs**.

SEE ALSO

m_fork(3PPS), m_kill_procs(3PPS)

NAME

m_single, m_multi – mark single-process code section

SYNOPSIS

C syntax:

```
#include <parallel/microtask.h>
m_single();
(CODE)
m_multi();
```

FORTRAN syntax

```
subroutine m_single
subroutine m_multi
```

DESCRIPTION

The **m_single** routine causes child processes to spin at a barrier until the parent process has executed the code following the **m_single** call and called the **m_multi** routine. The child processes then resume execution at the source line after the **m_multi** call. These routines are typically used to allow the parent process to perform I/O or other serial operations during an m_fork call.

NOTES

Calls to **m_single** are allowed only during **m_fork** calls. To suspend child processes after an **m_fork** call, use **m_park_procs** and **m_rele_procs.**

Do not call **m_multi** without calling **m_single first.**

Nested **m_single** calls are not allowed. Neither are **m_single** calls nested between calls to other lock routines, such as **s_lock** and **s_unlock.**

ERRORS

These routines can return the following error:

[EINVAL] The routine was called in an inappropriate context. For example, **m_fork** had never been called, or the processes were already suspended with a call to **m_park_procs.**

SEE ALSO

m_park_procs(3PPS)

NAME

m_sync – check in at barrier

SYNOPSIS

C syntax:

#include <parallel/microtask.h>
m_sync();

FORTRAN syntax

subroutine m_sync

DESCRIPTION

The **m_sync** routine causes a process to spin until all cooperating processes have reached the same point and called **m_sync**. The program must not call the **m_sync** routine unless there are multiple processes executing; that is, unless the program is executing a subprogram during an **m_fork** call and is not between a pair of **m_single/m_multi**, **m_lock/m_unlock**, or **s_lock/s_unlock** calls.

NOTES

Calls to **m_sync** reset the global counter (see **m_next(3PPS)**).

ERRORS

This routine fails if the following occurs:

[EINVAL] There are no child processes executing.

SEE ALSO

m_set_procs(3PPS), m_fork(3PPS), m_single(3PPS), m_park_procs(3PPS), m_kill_procs(3PPS), m_next(3PPS)

NAME

s_init_lock, s_lock, s_clock, s_unlock – initialize, lock, unlock locks

SYNOPSIS

C syntax:

```
#include <parallel/parallel.h>
slock_t *lp;
s_init_lock (lp);
S_INIT_LOCK (lp);
s_lock (lp);
S_LOCK (lp);
s_clock (lp);
S_CLOCK (lp);
s_unlock (lp);
S_UNLOCK (lp);
```

FORTRAN syntax

```
subroutine s_init_lock(lp)
subroutine s_lock(lp)
subroutine s_clock(lp)
subroutine s_unlock(lp)
integer*1 lp
```

DESCRIPTION

s_init_lock initializes a memory-based lock. After the lock is initialized, it can be locked with the **s_lock** or **s_clock** routine and unlocked with the **s_unlock** routine. There is no practical limit to the number of locks that can be used by a process.

In the C language, a lock is a shared data structure of type slock_t, as shown in the following declaration statement:

```
shared slock_t lock;
```

In FORTRAN, a lock is an INTEGER*1 variable. A FORTRAN lock must be placed in shared memory by declaring it in a common block using the FORTRAN compiler **–mp** option, which places all variables into shared memory.

s_lock and **s_clock** lock the lock whose address is **lp**. The lock must previously have been initialized using **s_init_lock**. **s_lock** is always successful; it spins as long as is necessary to acquire the lock. **s_clock** is successful only if the lock is free; if the lock is held by another process, **s_clock** returns the value L_FAILED. **s_clock** can be used when a process does not need to acquire a particular lock (for instance, when another lock could be used instead).

s_unlock unlocks the lock whose address is **lp**.

S_INIT_LOCK, S_LOCK, S_UNLOCK, and S_CLOCK are C-preprocessor macros. These macros are found in the header file */usr/include/parallel/parallel.h*. The macros are faster than the normal function calls, but they can add to the code size. See the source code in *<parallel/parallel.h>* for more information on the macros.

SEE ALSO

intro(3PPS), shmalloc(3PPS), fortran(1), ld(1)

NOTES

The function names **s_init_lock**, **s_lock**, **s_clock**, and **s_unlock** are used in C, Pascal, and FORTRAN. In C, the **lp** argument is passed as a pointer to the lock, while in FORTRAN, the argument is the address of the lock itself.

NAME

s_init_barrier, s_wait_barrier – initialize barrier, wait at barrier

SYNOPSIS

C syntax:

#include <parallel/parallel.h>
s_init_barrier (bp, nprocs);
sbarrier_t *bp;
int nprocs;

S_INIT_BARRIER (bp, nprocs);

s_wait_barrier (bp);
sbarrier_t *bp;

S_WAIT_BARRIER (bp);

FORTRAN syntax

integer*4 barrier, nprocs
subroutine s_init_barrier (barrier, nprocs)
subroutine s_wait_barrier (barrier)

DESCRIPTION

S_init_barrier initializes a barrier as a rendezvous point for exactly **nprocs** processes. This barrier can be used subsequently with **s_wait_barrier .**

In C, a barrier is a shared data structure of type sbarrier_t as shown in the following declaration statement:

```
shared sbarrier_t barrier;
```

In FORTRAN, a barrier is an INTEGER*4 variable. A FORTRAN barrier must be placed in shared memory either by declaring it in a common block and using the FORTRAN **–mp** option, which places all variables into shared memory.

s_wait_barrier delays the process in a busy wait until exactly **nprocs** processes have called **s_wait_barrier.** At that point, all processes exit the busy wait simultaneously. The barrier must have been previously initialized using **s_init_barrier .**

Results are undefined if more than **nprocs** processes call **s_wait_barrier.** A barrier can be used any number of times without being re-initialized. A barrier should not be re-initialized while processes are waiting at the barrier.

S_INIT_BARRIER and S_WAIT_BARRIER are C-preprocessor macros. These macros are found in the header file */usr/include/parallel/parallel.h.* When compiled in line, the macros are faster than the normal function calls, but they can add to the code size. See the source code in <parallel/parallel.h> for more information on the macros.

SEE ALSO

intro(3PPS), shmalloc(3PPS)

NAME

shbrk, shsbrk – change shared data segment size

SYNOPSIS

C syntax:

```
#include <parallel/parallel.h>
caddr_t shbrk(addr)
caddr_t addr;

caddr_t shsbrk(incr)
int incr;
```

FORTRAN syntax

none

DESCRIPTION

For a set of parallel processes executing a single application, **shbrk** sets the system's idea of the lowest shared data segment location not used by the program (called the shared break) to **addr** (rounded up to the next multiple of the system's page size). Locations greater than **addr** and below the stack pointer or another memory mapped region (see *mmap*(2SEQ)) are not in the address space and will thus cause a memory violation if accessed.

In the alternate function, **shsbrk**, **incr** more bytes are added to the program's shared data space and a pointer to the start of the new area is returned.

When a program begins execution via **execve**, the shared break is set at the highest location defined by the program. Ordinarily, therefore, only programs with growing shared data areas need to use **shsbrk**.

When **shbrk** and **shsbrk** increase a program's address space, they also attempt to increase its allowable resident set size (see **vm_ctl**(2SEQ)).

RETURN VALUE

Zero is returned if the shared break could be set; –1 if the program requests more memory than the system limit. **shsbrk** returns –1 if the break could not be set.

ERRORS

shbrk and **shsbrk** fails and no additional memory is allocated if any of the following error conditions occur:

[EINVAL] The shared break address would be lowered.

[ENOMEM] The routine cannot allocate file system space to expand the file which has been memory mapped to the shared data segment (see **mmap**(2SEQ)). The routine tries to allocate file system space from directories in the following order:

1. the directory indicated by the **$TMPPATH** environment variable
2. the current directory
3. the user's home directory
4. the directory */usr/tmp*
5. the directory */tmp*

[ENOMEM] The new shared data segment would overlap the stack segment.

SEE ALSO

execve(2), shmalloc(3PPS), end(3), vm_ctl(2SEQ), mmap(2SEQ)

BUGS

At this time, the size of the shared data segment can only be increased.

The shared break cannot be set above the stack segment limit for any of the processes in the program. (Remember that each process can have a different stack size.) If a process sets the shared break above the bottom of any process's stack, any reference to the overlapping area of that stack causes a core dump and aborts the program with a status of SIGSEGV.

shbrk and **shsbrk** use the SIGSEGV signal and signal handler for internal purposes. Users who declare their own SIGSEGV handler cannot expect reliable results from these routines.

If a **shbrk** or **shsbrk** call causes the shared data segment to overlap a memory mapped region, the shared data segment replaces the mapped region.

NAME

shmalloc, shrealloc, shfree – shared memory allocator

SYNOPSIS

C syntax:

```
char *shmalloc(size)
unsigned size;
shfree(ptr)
char *ptr;

char *shrealloc(prt,size)
char *ptr;
unsigned size;
```

FORTRAN syntax

none

DESCRIPTION

shmalloc, **shrealloc**, and **shfree** provide a simple general-purpose shared memory allocation package for a set of processes executing a single application. To use these routines, the program must have been linked with the Parallel Programming Library and *libseq.a*. **shmalloc** returns a pointer to a block of at least **size** bytes beginning on a 4-byte word boundary.

The argument to **shfree** is a pointer to a block previously allocated by **shmalloc**; this space is made available for further allocation, but its contents are left undisturbed.

Grave disorder results if the space assigned by **shmalloc** is overrun or if some random number is handed to **shfree**.

shmalloc maintains multiple lists of free blocks according to size, allocating space from the appropriate list. It calls **shsbrk** (see **shbrk**(3PPS)) to get more memory from the system when there is no suitable space already free. **shmalloc** and **shfree** coordinate the allocation of shared memory among the processes in the task. They maintain a consistent list of free blocks even when several processes are allocating shared memory concurrently. Concurrent requests for shared memory blocks always return unique blocks from the program's shared data segment.

shrealloc changes the size of the block pointed to by **ptr** to **size** bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If **ptr** points to a freed block, **shrealloc** returns NULL.

When any of these allocation routines increases a program's address space, it also attempts to increase its allowable resident set size (see **vm_ctl**(2SEQ)).

DIAGNOSTICS

shmalloc returns a NULL pointer (0) if there is no available shared memory or if the region has been detectably corrupted by storing data outside the bounds of a block. **shmalloc** may be recompiled to check the arena very stringently on every transaction; those sites with a source code license may check the source code to see how this can be done.

SEE ALSO

shbrk(3PPS), vm_ctl(2SEQ)

NAME

ctermid – generate filename for terminal

SYNOPSIS

include <stdio.h>
char *ctermid (s)
char *s;

DESCRIPTION

ctermid generates the pathname of the controlling terminal for the current process, and stores it in a string.

If **s** is a **NULL** pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to **ctermid**, and the address of which is returned. Otherwise, **s** is assumed to point to a character array of at least **L_ctermid** elements; the pathname is placed in this array and the value of **s** is returned. The constant **L_ctermid** is defined in the **<stdio.h>** header file.

NOTES

The difference between **ctermid** and **ttyname**(3C) is that **ttyname** must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while **ctermid** returns a string (*/dev/tty*) that will refer to the terminal if used as a filename. Thus **ttyname** is useful only if the process already has at least one file open to a terminal.

SEE ALSO

ttyname(3C)

NAME

cuserid – get character login name of the user

SYNOPSIS

```
#include <stdio.h>

char *cuserid (s)
char *s;
```

DESCRIPTION

cuserid generates a character-string representation of the login name that the owner of the current process is logged in under. If **s** is a NULL pointer, this representation is generated in an internal static area, the address of which is returned. Otherwise, **s** is assumed to point to an array of at least **L_cuserid** characters; the representation is left in this array. The constant **L_cuserid** is defined in the *<stdio.h>* header file.

DIAGNOSTICS

If the login name cannot be found, **cuserid** returns a NULL pointer; if **s** is not a NULL pointer, a null character **(\0)** will be placed at **s[0]**.

SEE ALSO

getlogin(3C), getpwent(3C).

NAME

fclose, fflush – close or flush a stream

SYNOPSIS

```
#include <stdio.h>

int fclose (stream)
FILE *stream;

int fflush (stream)
FILE *stream;
```

DESCRIPTION

fclose causes any buffered data for the named **stream** to be written out, and the **stream** to be closed.

fclose is performed automatically for all open files upon calling **exit**(2).

fflush causes any buffered data for the named **stream** to be written to that file. The **stream** remains open.

SEE ALSO

close(2), exit(2), fopen(3S), setbuf(3S), stdio(3S)

DIAGNOSTICS

These functions return 0 for success, and **EOF** if any error (such as trying to write to a file that has not been opened for writing) was detected.

NAME

ferror, feof, clearerr, fileno – stream status inquiries

SYNOPSIS

```
#include <stdio.h>

int ferror (stream)
FILE *stream;

int feof (stream)
FILE *stream;

void clearerr (stream)
FILE *stream;

int fileno (stream)
FILE *stream;
```

DESCRIPTION

ferror returns non-zero when an I/O error has previously occurred reading from or writing to the named **stream**, otherwise zero.

feof returns non-zero when **EOF** has previously been detected reading the named input **stream**, otherwise zero.

clearerr resets the error indicator and **EOF** indicator to zero on the named **stream**.

fileno returns the integer file descriptor associated with the named **stream**; see **open**(2).

NOTES

All these functions are implemented as macros; they cannot be declared or redeclared.

SEE ALSO

open(2), fopen(3S), stdio(3S)

NAME

fopen, freopen, fdopen – open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen (filename, type)
char *filename, *type;

FILE *freopen (filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen (fildes, type)
int fildes;
char *type;
```

DESCRIPTION

fopen opens the file named by **filename** and associates a **stream** with it. **fopen** returns a pointer to the **FILE** structure associated with the **stream.**

filename points to a character string that contains the name of the file to be opened.

Type is a character string having one of the following values:

r	open for reading
w	truncate or create for writing
a	append; open for writing at end of file, or create for writing
r+	open for update (reading and writing)
w+	truncate or create for update
a+	append; open or create for update at end-of-file

freopen substitutes the named file in place of the open **stream.** The original **stream** is closed, regardless of whether the open ultimately succeeds. **freopen** returns a pointer to the FILE structure associated with **stream.**

freopen is typically used to attach the preopened **streams** associated with **stdin, stdout** and **stderr** to other files.

fdopen associates a **stream** with a file descriptor. File descriptors are obtained from **open, dup, creat,** or **pipe**(2), which open files but do not return pointers to a FILE structure **stream.** Streams are necessary input for many of the Section 3S library routines. The **type** of **stream** must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting **stream.** However, output may not be directly followed by input without an intervening **fseek** or **rewind,** and input may not be directly followed by output without an intervening **fseek, rewind,** or an input operation which encounters end-of-file.

When a file is opened for append (i.e., when **type** is **a** or **a+**), it is impossible to overwrite information already in the file. **fseek** may be used to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If

two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

SEE ALSO

creat(2), dup(2), open(2), pipe(2), fclose(3S), fseek(3S), stdio(3S), getdtablesize(2seq), setdtablesize(2seq), getdtablemax(2seq)

DIAGNOSTICS

fopen, fdopen, and **freopen** return a **NULL** pointer on failure.

ERRORS

[ENOMEM]	There is not enough available memory to allocate additional FILE structures.
[EMFILE]	Too many files are open.

NAME

fread, fwrite – binary input/output

SYNOPSIS

```
#include <stdio.h>
#include <sys/types.h>

int fread (ptr, size, nitems, stream)
char *ptr;
int nitems;
size_t size;
FILE *stream;

int fwrite (ptr, size, nitems, stream)
char *ptr;
int nitems;
size_t size;
FILE *stream;
```

DESCRIPTION

fread copies, into an array pointed to by **ptr**, **nitems** items of data from the named input **stream**, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length **size**. **fread** stops appending bytes if an end-of-file or error condition is encountered while reading **stream**, or if **nitems** items have been read. **fread** leaves the file pointer in **stream**, if defined, pointing to the byte following the last byte read if there is one. **fread** does not change the contents of **stream**.

fwrite appends at most **nitems** items of data from the array pointed to by **ptr** to the named output **stream**. **fwrite** stops appending when it has appended **nitems** items of data or if an error condition is encountered on **stream**. **fwrite** does not change the contents of the array pointed to by **ptr**.

The argument **size** is typically **sizeof(*ptr)** where the pseudofunction **sizeof** specifies the length of an item pointed to by **ptr**. If **ptr** points to a data type other than **char** it should be cast into a pointer to **char**.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S), stdio(3S)

DIAGNOSTICS

fread and **fwrite** return the number of items read or written. If **nitems** is non-positive, no characters are read or written and 0 is returned by both **fread** and **fwrite**.

NAME

fseek, rewind, ftell – reposition a file pointer in a stream

SYNOPSIS

```
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind (stream)
FILE *stream;

long ftell (stream)
FILE *stream;
```

DESCRIPTION

fseek sets the position of the next input or output operation on the **stream**. The new position is at the signed distance **offset** bytes from the beginning, from the current position, or from the end of the file, according as **ptrname** has the value 0, 1, or 2.

rewind (stream) is equivalent to **fseek(stream,0,0)**, except that no value is returned.

fseek and **rewind** undo any effects of **ungetc**(3S).

After **fseek** or **rewind**, the next operation on a file opened for update may be either input or output.

ftell returns the offset of the current byte relative to the beginning of the file associated with the named **stream**.

SEE ALSO

lseek(2), fopen(3S), popen(3S), stdio(3S), ungetc(3S)

DIAGNOSTICS

fseek returns non-zero for improper seeks, otherwise zero. An improper seek can be, for example, an **fseek** done on a file that has not been opened via **fopen**; in particular, **fseek** may not be used on a terminal, or on a file opened via **popen**(3S).

WARNINGS

Although on the DYNIX/ptx system an offset returned by **ftell** is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to non-UNIX systems requires that an offset be used by **fseek** directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.

NAME

getc, getchar, fgetc, getw – get character or word from a stream

SYNOPSIS

```
#include <stdio.h>

int getc (stream)
FILE *stream;

int getchar ()

int fgetc (stream)
FILE *stream;

int getw (stream)
FILE *stream;
```

DESCRIPTION

getc returns the next character (i.e., byte) from the named input **stream**, as an integer. It also moves the file pointer, if defined, ahead one character in **stream**. **getchar** is defined as **getc(stdin)**. **getc** and **getchar** are macros.

fgetc behaves like **getc**, but is a function rather than a macro. **fgetc** runs more slowly than **getc**, but it takes less space per invocation and its name can be passed as an argument to a function.

getw returns the next word (i.e., integer) from the named input **stream**. **getw** increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. **getw** assumes no special alignment in the file.

SEE ALSO

fclose(3S), ferror(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S), stdio(3S)

DIAGNOSTICS

These functions return the constant **EOF** at end-of-file or upon an error. Because **EOF** is a valid integer, **ferror**(3S) should be used to detect **getw** errors.

WARNINGS

If the integer value returned by **getc**, **getchar**, or **fgetc** is stored into a character variable and then compared against the integer constant **EOF**, the comparison may never succeed, because sign-extension of a character on widening to integer is machine-dependent.

Because it is implemented as a macro, **getc** evaluates a **stream** argument more than once. In particular, **getc(*f++)** does not work sensibly. **fgetc** should be used instead.

Because of possible differences in word length and byte ordering, files written using **putw** are machine-dependent, and may not be read using **getw** on a different processor.

NAME

gets, fgets – get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets (s)
char *s;

char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

DESCRIPTION

gets reads characters from the standard input stream, **stdin**, into the array pointed to by **s**, until a new-line character is read or an end-of-file condition is encountered. The new-line character is discarded and the string is terminated with a null character.

fgets reads characters from the **stream** into the array pointed to by **s**, until **n**–1 characters are read, or a new-line character is read and transferred to **s**, or an end-of-file condition is encountered. The string is then terminated with a null character.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S), stdio(3S)

DIAGNOSTICS

If end-of-file is encountered and no characters have been read, no characters are transferred to **s** and a **NULL** pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a **NULL** pointer is returned. Otherwise **s** is returned.

NAME

popen, pclose – initiate pipe to/from a process

SYNOPSIS

#include <stdio.h>

FILE *popen (command, type)
char *command, *type;

int pclose (stream)
FILE *stream;

DESCRIPTION

popen creates a pipe between the calling program and the command to be executed. The arguments to **popen** are pointers to null-terminated strings. **command** consists of a shell command line. **type** is an I/O mode, either **r** for reading or **w** for writing. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is **w**, by writing to the file **stream**; and one can read from the standard output of the command, if the I/O mode is **r**, by reading from the file **stream**.

A stream opened by **popen** should be closed by **pclose**, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type **r** command may be used as an input filter and a type **w** as an output filter.

EXAMPLE

A typical call may be:

```
char *cmd = "ls *.c";
FILE *ptr;
if ((ptr = popen(cmd, "r")) != NULL)
      while (fgets(buf, n, ptr) != NULL)
            (void) printf("%s ",buf);
```

This will print in **stdout** [see **stdio** (3S)] all the file names in the current directory that have a *.c* suffix.

SEE ALSO

pipe(2), wait(2), fclose(3S), fopen(3S), stdio(3S), system(3S).

DIAGNOSTICS

popen returns a **NULL** pointer if files or processes cannot be created.

pclose returns –1 if **stream** is not associated with a **popen**ed command.

WARNINGS

If the original and **popen**ed processes concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Problems with an output filter may be forestalled by careful buffer flushing, e.g. with **fflush** (see **fclose**(3S)).

NAME

printf, fprintf, sprintf – print formatted output

SYNOPSIS

```
#include <stdio.h>

int printf (format , arg ... )
char *format;

int fprintf (stream, format , arg ... )
FILE *stream;
char *format;

int sprintf (s, format [, arg ] ...)
char *s, *format;
```

DESCRIPTION

printf places output on the standard output stream **stdout**. **fprintf** places output on the named output **stream**. **sprintf** places "output," followed by the null character (**\0**), in consecutive bytes starting at ***s**; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the **\0** in the case of **sprintf**), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its **args** under control of the **format**. The **format** is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more **args**. The results are undefined if there are insufficient **args** for the format. If the format is exhausted while **args** remain, the excess **args** are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more **flags**, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum **field width**. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag '–', described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A **precision** that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion character applies to a long integer **arg**. An **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (*) instead of a digit string. In this case, an integer **arg** supplies the field width or precision. The **arg** that is actually converted is not fetched until the conversion letter is seen, so the **args** specifying field width or precision must appear *before* the **arg** (if any) to be converted. A negative field width argument is taken as a '–' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

– The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

This flag specifies that the value is to be converted to an alternate form. For **c**, **d**, **i**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x or X** conversion, a non-zero result will have **0x or 0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeros will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,i,o,u,x,X
The integer **arg** is converted to signed decimal (**d** or **i**), unsigned octal, (**o**), decimal (**u**), or hexadecimal notation (**x** or **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

f The float or double **arg** is converted to decimal notation in the style "[–]ddd.ddd," where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly 0, no decimal point appears.

e,E The float or double **arg** is converted in the style "[–]d.ddde±dd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is

missing, six digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

g,G The float or double **arg** is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** will be used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeros are removed from the result; a decimal point appears only if it is followed by a digit.

c The character **arg** is printed.

s The **arg** is taken to be a string (character pointer) and characters from the string are printed until a null character (**\0**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for **arg** will yield undefined results.

% Print a %; no argument is converted.

In printing floating-point types (float and double), if the exponent is 0x7FF and the mantissa is not equal to zero, then the output is

[-]NaN0xddddddddd

where 0xddddddddd is the hexadecimal representation of the leftmost 32 bits of the mantissa. If the mantissa is zero, the output is

[±]inf.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by **printf** and **fprintf** are printed as if **putc**(3S) had been called.

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02," where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

To print π to 5 decimal places:

```
printf("pi = %.5f", 4 * atan(1.0));
```

SEE ALSO

ecvt(3C), putc(3S), scanf(3S), stdio(3S)

NAME

putc, putchar, fputc, putw – put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc (c, stream)
int c;
FILE *stream;

int putchar (c)
int c;

int fputc (c, stream)
int c;
FILE *stream;

int putw (w, stream)
int w;
FILE *stream;
```

DESCRIPTION

putc writes the character **c** onto the output **stream** (at the position where the file pointer, if defined, is pointing). **putchar(c)** is defined as **putc(c,stdout)**. **putc** and **putchar** are macros.

fputc behaves like **putc**, but is a function rather than a macro. **fputc** runs more slowly than **putc**, but it takes less space per invocation and its name can be passed as an argument to a function.

putw writes the word (i.e. integer) **w** to the output **stream** (at the position at which the file pointer, if defined, is pointing). The size of a word is the size of an integer and varies from machine to machine. **putw** neither assumes nor causes special alignment in the file.

SEE ALSO

fclose(3S), ferror(3S), fopen(3S), fread(3S), printf(3S), puts(3S), setbuf(3S), stdio(3S)

DIAGNOSTICS

On success, these functions (with the exception of **putw**) each return the value they have written. [**Putw** returns **ferror (stream)**]. On failure, they return the constant **EOF**. This occurs if the file **stream** is not open for writing or if the output file cannot grow. Because **EOF** is a valid integer, **ferror**(3S) should be used to detect **putw** errors.

WARNINGS

Because it is implemented as a macro, **putc** evaluates a **stream** argument more than once. In particular, `putc(c, *f++);` doesn't work sensibly. **fputc** should be used instead.

Because of possible differences in word length and byte ordering, files written using **putw** are machine-dependent, and may not be read using **getw** on a different processor.

NAME

puts, fputs – put a string on a stream

SYNOPSIS

```
#include <stdio.h>

int puts (s)
char *s;

int fputs (s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

puts writes the null-terminated string pointed to by **s**, followed by a new-line character, to the standard output **stream stdout**.

fputs writes the null-terminated string pointed to by **s** to the named output **stream**.

Neither function writes the terminating null character.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), printf(3S), putc(3S), stdio(3S)

DIAGNOSTICS

Both routines return **EOF** on error. This will happen if the routines try to write on a file that has not been opened for writing.

NOTES

puts appends a new-line character while **fputs** does not.

NAME

scanf, fscanf, sscanf – convert formatted input

SYNOPSIS

```
#include <stdio.h>

int scanf (format [ , pointer ] ... )
char *format;

int fscanf (stream, format [ , pointer ] ... )
FILE *stream;
char *format;

int sscanf (s, format [ , pointer ] ... )
char *s, *format;
```

DESCRIPTION

scanf reads from the standard input stream **stdin**. **fscanf** reads from the named input **stream**. **Sscanf** reads from the character string **s**. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string **format** described below, and a set of **pointer** arguments indicating where the converted input should be stored. The results are undefined in there are insufficient **args** for the format. If the format is exhausted while **args** remain, the excess **args** are simply ignored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, cause input to be read up to the next non-white-space character.

2. An ordinary character (not %), which must match the next character of the input stream.

3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion codes are legal:

% a single % is expected in the input at this point; no assignment is done.

d a decimal integer is expected; the corresponding argument should be an integer pointer.

u an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.

o an octal integer is expected; the corresponding argument should be an integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

i an integer is expected; the corresponding argument should be an integer pointer. It will store the value of the next input item interpreted according to C conventions: a leading "0" implies octal; a leading "0x" implies hexadecimal; otherwise, decimal.

n stores in an integer argument the total number of characters (including white space) that have been scanned so far since the function call. No input is consumed.

e,f,g a floating-point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a **float**. The input format for floating-point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an **E** or an **e**, followed by an optional +, –, or space, followed by an integer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white-space character.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use **%1s**. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.

[indicates string data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters **not** contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first–last*, thus [0123456789] may be expressed [0–9]. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself

whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating **\0**, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, **x** and **i** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The **l** or **h** modifier is ignored for other conversion characters.

scanf conversion terminates at **EOF**, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, **EOF** is returned.

EXAMPLES

The call:

```
int n ; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to **n** the value **3**, to **i** the value **25**, to **x** the value **5.432**, and **name** will contain **thompson\0** . Or:

```
int i, j; float x; char name[50];
(void) scanf("%i%2d%f%*d %[0-9] ", &j, &i, &x, name);
```

with input:

```
011 56789 0123 56a72
```

will assign **9** to **j**, **56** to **i**, **789.0** to **x**, skip **0123**, and place the string **56\0** in **name**. The next call to **getchar** [see **getc**(3S)] will return **a**. Or:

```
int i, j, s, e; char name[50];
(void) scanf("%i %i %n%s%n", &i, &j, &s, name, &e);
```

with input:

```
0x11 0xy johnson
```

will assign **17** to **i**, **0** to **j**, **6** to **s**, will place the string **xy\0** in **name**, and will assign **8** to **e**. Thus, the length of **name** is **e - s = 2**. The next call to **getchar** [see **getc**(3S)] will return a blank.

SEE ALSO

getc(3S), printf(3S), stdio(3S), strtod(3C), strtol(3C)

DIAGNOSTICS

These functions return **EOF** on end of input and a short count for missing or illegal data items.

WARNINGS

Trailing white space (including a new-line) is left unread unless matched in the control string.

NAME

setbuf, setvbuf – assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

void setbuf (stream, buf)
FILE *stream;
char *buf;

int setvbuf (stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

DESCRIPTION

setbuf may be used after a stream has been opened but before it is read or written. It causes the array pointed to by **buf** to be used instead of an automatically allocated buffer. If **buf** is the NULL pointer input/output will be completely unbuffered.

A constant **BUFSIZ**, defined in the **<stdio.h>** header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

setvbuf may be used after a stream has been opened but before it is read or written. **type** determines how **stream** will be buffered. Legal values for **type** (defined in **<stdio.h>**) are:

_IOFBF	causes input/output to be fully buffered.
_IOLBF	causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.
_IONBF	causes input/output to be completely unbuffered.

If **buf** is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. **size** specifies the size of the buffer to be used. The constant **BUFSIZ** in **<stdio.h>** is suggested as a good buffer size. If input/output is unbuffered, **buf** and **size** are ignored.

By default, output to a terminal is line buffered and all other input/output is fully buffered.

SEE ALSO

fopen(3S), getc(3S), malloc(3C), putc(3S), stdio(3S)

DIAGNOSTICS

If an illegal value for **type** or **size** is provided, **setvbuf** returns a non-zero value. Otherwise, the value returned will be zero.

NOTES

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

NAME

stdio – standard buffered input/output package

SYNOPSIS

#include <stdio.h>

FILE *stdin, *stdout, *stderr;

DESCRIPTION

The functions described in the entries of sub-class 3S of this manual constitute an efficient, user-level I/O buffering scheme. The in-line macros **getc**(3S) and **putc**(3S) handle characters quickly. The macros **getchar** and **putchar**, and the higher-level routines **fgetc**, **fgets**, **fprintf**, **fputc**, **fputs**, **fread**, **fscanf**, **fwrite**, **gets**, **getw**, **printf**, **puts**, **putw**, and **scanf** all use or act as if they use **getc** and **putc**; they can be freely intermixed.

A file with associated buffering is called a **stream** and is declared to be a pointer to a defined type **FILE**. **fopen**(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the <stdio.h> header file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant **NULL** (0) designates a nonexistent pointer.

An integer-constant **EOF** (–1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

An integer constant **BUFSIZ** specifies the size of the buffers used by the particular implementation.

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in the entries of sub-class 3S of this manual are declared in that header file and need no further declaration. The constants and the following "functions" are implemented as macros (redeclaration of these names is perilous): **getc**, **getchar**, **putc**, **putchar**, **ferror**, **feof**, **clearerr**, and **fileno**.

Output streams, with the exception of the standard error stream **stderr**, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream **stderr** is by default unbuffered, but use of **freopen** [see **fopen**(3S)] will cause it to become buffered or line-buffered. When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block. When it is

line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested). **setbuf**(3S) or **setvbuf**() in **setbuf**(3S) may be used to change the stream's buffering strategy.

SEE ALSO

open(2), close(2), lseek(2), pipe(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), getc(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S)

DIAGNOSTICS

Invalid **stream** pointers will usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.

NAME

system – issue a shell command

SYNOPSIS

```
#include <stdio.h>

int system (string)
char *string;
```

DESCRIPTION

system causes the **string** to be given to **sh**(1) as input, as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

FILES

/bin/sh

SEE ALSO

sh(1), exec(2)

DIAGNOSTICS

system forks to create a child process that in turn exec's */bin/sh* in order to execute **string**. If the fork or exec fails, **system** returns a negative value and sets **errno**.

NAME

tmpfile – create a temporary file

SYNOPSIS

#include <stdio.h>

FILE *tmpfile ()

DESCRIPTION

tmpfile creates a temporary file using a name generated by **tmpnam**(3S), and returns a corresponding **FILE** pointer. If the file cannot be opened, an error message is printed using **perror**(3C), and a **NULL** pointer is returned. The file will automatically be deleted when the process using it terminates. The file is opened for update (**w+**).

SEE ALSO

creat(2), unlink(2), fopen(3S), mktemp(3C), perror(3C), stdio(3S), tmpnam(3S)

NAME

tmpnam, tempnam – create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam (s)
char *s;

char *tempnam (dir, pfx)
char *dir, *pfx;
```

DESCRIPTION

These functions generate filenames that can safely be used for a temporary file.

tmpnam always generates a filename using the path-prefix defined as **P_tmpdir** in the **<stdio.h>** header file. If **s** is **NULL**, **tmpnam** leaves its result in an internal static area and returns a pointer to that area. The next call to **tmpnam** will destroy the contents of the area. If **s** is not NULL, it is assumed to be the address of an array of at least **L_tmpnam** bytes, where **L_tmpnam** is a constant defined in **<stdio.h>**; **tmpnam** places its result in that array and returns **s**.

tempnam allows the user to control the choice of a directory. The argument **dir** points to the name of the directory in which the file is to be created. If **dir** is NULL or points to a string that is not a name for an appropriate directory, the path-prefix defined as **P_tmpdir** in the **<stdio.h>** header file is used. If that directory is not accessible, **tmp** will be used as a last resort. This entire sequence can be up-staged by providing an environment variable TMPDIR in the user's environment, whose value is the name of the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the **pfx** argument for this. This argument may be **NULL** or point to a string of up to five characters to be used as the first few characters of the temporary-filename.

tempnam uses **malloc**(3C) to get space for the constructed filename, and returns a pointer to this area. Thus, any pointer value returned from **tempnam** may serve as an argument to **free** [see **malloc**(3C)]. If **tempnam** cannot return the expected result for any reason, i.e. **malloc**(3C) failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.

NOTES

These functions generate a different filename each time they are called.

Files created using these functions and either **fopen**(3S) or **creat**(2) are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use **unlink**(2) to remove the file when its use is ended.

SEE ALSO

creat(2), unlink(2), fopen(3S), malloc(3C), mktemp(3C), tmpfile(3S)

WARNINGS

If called more than **TMP_MAX** times in a single process, these functions will

start recycling previously used names. (**TMP_MAX** is a constant defined in **<stdio.h>**.)

Between the time a filename is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or **mktemp**, and the filenames are chosen to render duplication by other means unlikely.

NAME

ungetc – push character back into input stream

SYNOPSIS

```
#include <stdio.h>

int ungetc (c, stream)
int c;
FILE *stream;
```

DESCRIPTION

ungetc inserts the character **c** into the buffer associated with an input **stream**. That character, **c**, will be returned by the next **getc**(3S) call on that **stream**. **ungetc** returns **c**, and leaves the file **stream** unchanged.

One character of pushback is guaranteed, provided something has already been read from the stream and the stream is actually buffered.

If **c** equals **EOF**, **ungetc** does nothing to the buffer and returns **EOF**.

fseek(3S) erases all memory of inserted characters.

SEE ALSO

fseek(3S), getc(3S), setbuf(3S), stdio(3S)

DIAGNOSTICS

ungetc returns **EOF** if it cannot insert the character.

BUGS

When **stream** is **stdin**, one character may be pushed back onto the buffer without a previous read statement.

NAME

vprintf, vfprintf, vsprintf – print formatted output of a varargs argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf (format, ap)
char *format;
va_list ap;

int vfprintf (stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf (s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

vprintf, **vfprintf**, and **vsprintf** are the same as **printf**, **fprintf**, and **sprintf** respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **varargs**(5).

EXAMPLE

The following demonstrates the use of **vfprintf** to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
      .
      .
 *    error should be called like
 *         error(function_name, format, arg1, arg2...);   */
/*VARARGS*/

void
error(va_alist)

/* Note that the function_name and format arguments cannot be
 * separately declared because of the definition of varargs. */
va_dcl
{
      va_list args;
      char *fmt;

      va_start(args);
      /* print out name of function causing error */
      (void)fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
      fmt = va_arg(args, char *);
      /* print out remainder of message */
      (void)vfprintf(stderr, fmt, args);
      va_end(args);
      (void)abort( );
}
```

SEE ALSO

printf(3S), varargs(5)

NAME

dis386 – disassemble Series 386 instructions

SYNOPSIS

```
char *dis386(addr, len_p)
unsigned long addr;
int *len_p;

dis386_init(iread, dolook, do_lnums, xlate_name,
    xlate_local, oflag, Rflag, sflag)
unsigned char (*iread)();
void (*dolook)();
void (*do_lnums)();
char *(*xlate_name)();
void (*xlate_local)();
int oflag, Rflag, sflag;
```

DESCRIPTION

dis386 is a disassembler. **dis386** returns a pointer to a string containing the disassembled instruction. **dis386** in initialized with **dis386_init.**

dis386 takes the following arguments:

addr The of the address of the first byte of the instruction to be disassembled.

len_p A pointer to an integer. **dis386** stores the length of the string containing the disassembled instruction into the integer.

dis386_init takes the following arguments:

iread A pointer to a routine to return actual bytes to be disassembled from the instruction space. *iread* takes an unsigned argument that is the address of the byte to be read. The first call passed this address to **dis386.** Subsequent calls increase the address by 1.

dolook Used to disassamble jump tables. *dolook* gives the calling program the address at which **dis386** is disassembling so that the calling program can print the label if appropriate. *dolook* takes an unsigned argument that specifies where the label might be.

do_lnums Used to disassemble jump tables. *do_lnums* gives the calling program the line numbers at which **dis386** is disassembling so that the calling program can print the line numbers if appropriate. *do_lnums* does not take any arguments.

xlate_name A pointer to a routine to return the symbolic name of an absolute address. If a **NULL** pointer is passed as the routine to translate addresses, no translation is done. *xlate_name* takes an unsigned argument that specifies the symbol address.

xlate_local A pointer to a routine to return the symbolic name of local symbols, function arguments, and arrays. *xlate_local* can

take any of the following arguments: the offset from the base register (type long), the register number (type int), or a pointer to a pointer to a char (type char**).

oflag If *oflag* is 0, the **dis386** output is hexadecimal; if *oflag* is 1, the output is octal.

Rflag If *Rflag* is 0, 32-bit mode is specified; if *Rflag* is 1, 16-bit mode is specified.

sflag A value of 0 specifies no symbolic diassembly; a value of 1 specifies symbolic diassembly, unless either *xlate_name* or *xlate_local* are not **NULL**.

The routine to return bytes from the instruction space should be declared as follows:

```
unsigned char iread(addr)
   unsigned int addr;
```

It should return the byte at the address in the instruction space given by **addr**.

The routine to return the symbolic address of an absolute address should be declared as follows:

```
char *xlate_name(addr)
   unsigned int addr;
```

It should return a pointer to a static buffer holding the symbolic name of the location given by **addr**. If no name is found, a **NULL** pointer should be returned.

The routine to return the symbolic name of a local variable should be declared as follows:

```
void xlate_local(val, regno, sp)
     long val;
     int regno;
     char **sp;
```

It should write into the string pointed to by sp the symbolic name of the local variable with offset val from register regno. If no name is found, a NULL pointer should be returned.

dis386 returns a pointer to a static character string that is the disassembly of the given address, and also stores the length of the instruction in the integer pointed to by **len_p**.

This routine is found in the Sequent library, designated in **ld**(1) or **cc**(1) by **-lseq**.

SEE ALSO

dis(1)

NOTES

This is a Sequent library routine. To use it, specify the **-lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

NAME

getgeombyname – get disk geometry by name

SYNOPSIS

```
#include <sys/zdc.h>
#include <diskinfo.h>

struct geomtab *
getgeombyname(disktype)
char *disktype;
```

DESCRIPTION

getgeombyname takes a disk type name (e.g., *m2333k*) and returns a structure describing its geometry information. All information is obtained from files in the directory */etc/diskinfo* which end in the suffix *.geom*. Each of these files contains the geometry information for one disk type. See */usr/include/diskinfo.h* for a description of the fields in *struct geomtab*.

SEE ALSO

diskinfo(4)

NOTES

This is a Sequent library routine. To use it, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

NAME

addmntent, endmntent, getmntent, hasmntopt, setmntent – get file system descriptor file entry

SYNOPSIS

```
#include <stdio.h>
#include <sys/types.h>
#include <mntent.h>

int addmntent(filep, mnt)
FILE *filep;
struct mntent *mnt;

int endmntent(filep)
FILE *filep;

char *hasmntopt(mnt, opt)
struct mntent *mnt;
char *opt;

struct mntent *getmntent(filep)
FILE *filep;

FILE *setmntent(filep, type)
char *filep;
char *type;
```

DESCRIPTION

These routines are used to access the file system description file */etc/fstab*. They are also used to access and update the mounted file system description file */etc/mnttab*.

setmntent opens a file system description file and returns a file pointer which can then be used with **getmntent**, **addmntent**, or **endmntent**. The **type** argument is the same as in **fopen**(3). **getmntent** reads the next line from **filep** and returns a pointer to an object with the following structure containing the broken-out fields of a line in the filesystem description file. The structure is defined in *<mntent.h>*. The fields have meanings described in *fstab*(4).

```
struct mntent {
        char    *mnt_fsname;  /* file system name */
        char    *mnt_dir;     /* file system path prefix */
        char    *mnt_type;    /* 4.2, nfs, or ignore */
        char    *mnt_opts;    /* ro, quota, etc. */
        time_t   mnt_time;    /* mount time (mnttab only) */
};
```

addmntent adds the **mntent** structure **mnt** to the end of the open file **filep**. Note that **filep** has to be opened for writing if this is to work. **hasmntopt** scans the **mnt_opts** field of the **mntent** structure **mnt** for a substring that matches **opt**. It returns the address of the substring if a match is found, 0 otherwise. **endmntent** closes the file.

FILES

/etc/fstab
/etc/mnttab

SEE ALSO

fstab(4), mnttab(4)

DIAGNOSTICS

A **NULL** pointer is returned on EOF or error.

NOTES

These are Sequent library routines. To use them, specify the **–lseq** option to link with the Sequent library. Since these are Sequent extensions, code using these routines may not be portable to other systems.

BUGS

The returned **mntent** structure points to static information that is overwritten in each call.

NAME

getpseudotty – get pseudo terminals

SYNOPSIS

int getpseudotty(slave, master)
char **slave, **master;

DESCRIPTION

getpseudotty locates an available pseudo terminal device pair and returns an open file descriptor of the master side opened for reading and writing. The pathname of the pair is returned through the slave and master arguments.

If **getpseudotty** is unable to find any available pseudo terminals, it returns –1.

FILES

/dev/getpty

SEE ALSO

streampty(7)

NOTES

This is a Sequent library routine. To use it, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

BUGS

The return values of **getpseudotty** point to static data whose contents are overwritten by each call.

NAME

getscsiinfo, getscsimatch – get scsiinfo structure

SYNOPSIS

```
#include <sys/zdc.h>
#include <diskinfo.h>

struct scsiinfo *
getscsimatch(vendor, product)
char *vendor, *product;

struct scsiinfo *
getscsiinfo()
```

DESCRIPTION

getscsimatch takes the vendor and product names of the disk as returned by the SCSI INQUIRY command and locates the *.scsi* (e.g., wren3.scsi) file in */etc/diskinfo* with matching vendor and product strings. It then returns the data in this file in a *scsiinfo* structure. **getscsiinfo** finds the next *.scsi* file in */etc/diskinfo* and creates and returns a *scsiinfo* structure built from that file. On error or when there are no more *.scsi* files in the directory, **NULL** is returned. This information is used by the **format** program. Note that the vendor string must be padded with blanks to be 8 characters long, and the product string must be padded to 16 characters, as this is the way the strings are returned by the SCSI INQUIRY command. See */usr/include/diskinfo.h* for a description of the fields in *struct scsiinfo*.

SEE ALSO

diskinfo(4) **format**(1M)

NOTES

These are Sequent library routines. To use them, specify the **-lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

NAME

getusclk, usclk_init – get/init microsecond clock

SYNOPSIS

#include <usclkc.h>
void usclk_init()

usclk_t getusclk()

usclk_t GETUSCLK()

DESCRIPTION

The Sequent microsecond clock (usclk) is a programmatically accessible 32-bit up-counter which increments in real (wall clock) time at a one microsecond per tick rate. It is reset to zero at power-up and free runs thereafter. There is no notification as the clock rolls over its maximum value and returns to zero.

usclk_init maps the microsecond clock into process address space, making it accessible for subsequent use. **usclk_init** should be called only once and must precede the first call to **getusclk** .

getusclk returns the current 32-bit microsecond clock value.

GETUSCLK is a **cpp** macro which accesses the microsecond clock without the overhead of a function call. For this reason this is the preferred method for referencing the microsecond clock.

The routines described here are in */usr/lib/libseq.a*. They are linked into a program using the **–lseq** option in the **cc** or **ld** command line. GETUSCLK is defined in the *usclkc.h* header file.

SEE ALSO

usclk(7)

DIAGNOSTICS

usclk_init fails with exit status 1 if it cannot open, map, or access the microsecond clock device.

NOTES

The mapping established by **usclk_init** is inherited by forked processes.

The call overhead for GETUSCLK is nil on Symmetry.

These are Sequent library routines. To use them, specify the **–lseq** option to link with the Sequent library. Since these are Sequent extensions, code using these routines may not be portable to other systems.

NAME

getzdinfo, getzdinfobyname – get zdinfo structure

SYNOPSIS

```
#include <sys/zdc.h>
#include <diskinfo.h>

struct zdinfo *
getzdinfobyname(disktype)
char *disktype;

struct zdinfo *
getzdinfo()
```

DESCRIPTION

getzdinfobyname takes a disk type name (e.g., m2333k) and returns a structure containing the disk type name and the ZDC channel configuration data for that disk type. All information is obtained from files in the directory */etc/diskinfo* which end in the suffix *.zd*. Each of these files contains the channel configuration data for one disk type. **getzdinfo** finds the next *'.zd'* file in */etc/diskinfo* and creates and returns a zdinfo structure built from that file. On error or when there are no more *.zd* files in the directory, **NULL** is returned. This information is provided primarily for use by the *format* program. See */usr/include/diskinfo.h* and */usr/include/sys/zdc.h* for a description of the fields in *struct zdinfo*.

SEE ALSO

diskinfo(4) **format**(1M)

NOTES

These are Sequent library routines. To use them, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

NAME

initgroups – initialize group access list

SYNOPSIS

initgroups(name)
char *name;

DESCRIPTION

initgroups reads through the group file and sets up, using the **setgroups**(2seq) call, the supplementary group access list for the user specified in **name**. Only the first NGROUPS_MAX [see **intro**(2)] entries are used if the user is in too many groups.

FILES

/etc/group

SEE ALSO

intro(2), setgroups(2seq)

DIAGNOSTICS

initgroups returns 1 if it was not invoked by the superuser and 0 if successful.

NOTES

This is a Sequent library routine. To use it, specify the **-lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

BUGS

initgroups uses the routines based on **getgrent**(3). If the invoking program uses any of these routines, the group structure is overwritten in the call to **initgroups**.

NAME

nblocks – calculate number of blocks and indirect blocks

SYNOPSIS

unsigned long nblocks(size, bsize, fsize)
unsigned long size, bsize, fsize;

DESCRIPTION

nblocks returns the total number of DEV_BSIZE blocks (typically 512 bytes) a file of **size** bytes would use on a filesystem. The total includes single indirect, double indirect and triple indirect blocks. The **bsize** parameter is the block size of the filesystem. The **fsize** parameter is currently ignored.

Partial blocks are counted in block-size chunks.

NOTES

This is a Sequent library routine. To use it, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine will not be portable to other systems.

BUGS

Large sparce files may return an incorrect number of blocks.

NAME

offline_all – take all but one active processor off line

SYNOPSIS

int offline_all()

DESCRIPTION

The **offline_all** routine takes all processors off line except the lowest-numbered active processor (normally processor 0).

SEE ALSO

tmp_ctl(2), tmp_affinity(2)

DIAGNOSTICS

Returns 1 if successful, 0 otherwise. If unsuccessful, use **tmp_ctl**(2) to get the status of the remaining processors.

NOTES

This is a Sequent library routine. To use it, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

NAME

read_constab – read constab entries into a structure

SYNOPSIS

#include <constab.h>

read_constab(table, last_entry)
struct constab **table;
struct constab **last_entry;

DESCRIPTION

The file */etc/constab* contains the results of virtual device bindings which result from invocations of the **devbuild**(1M) command. The contents of this file may be read into a convenient structure through the *read_constab()* routine. Upon successful completion of the routine, **table** points to a dynamically allocated table of `struct constab` structures, and **last_entry** points to the last entry of the table. See the file */usr/include/constab.h* for a definition of `struct constab`.

SEE ALSO

devbuild (1M), constab (4)

NOTES

This is a Sequent library routine. To use it, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

NAME

valloc – aligned memory allocator

SYNOPSIS

char *valloc(size)
unsigned size;

DESCRIPTION

valloc allocates **size** bytes aligned on a page boundary. It is implemented by calling **malloc**(3) with a slightly larger request, saving the true beginning of the block allocated, and returning a properly aligned pointer.

DIAGNOSTICS

valloc returns a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block.

NOTES

This is a Sequent library routine. To use it, specify the **–lseq** option to link with the Sequent library. Since this is a Sequent extension, code using this routine may not be portable to other systems.

BUGS

vfree isn't implemented.

(3X)

NAME

assert – verify program assertion

SYNOPSIS

#include <assert.h>

assert (expression)
int expression;

DESCRIPTION

This macro is useful for putting diagnostics into programs. When it is executed, if **expression** is false (zero), **assert** prints

`Assertion failed:` *expression*`, file` *xyz*`, line` *nnn*

on the standard error output and aborts. In the error message, *xyz* is the name of the source file and *nnn* the source line number of the **assert** statement.

Compiling with the preprocessor option **-DNDEBUG** [see **cpp**(1)], or with the preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement, will stop assertions from being compiled into the program.

SEE ALSO

cpp(1), abort(3C)

WARNINGS

Since **assert** is implemented as a macro, the **expression** may not contain any string literals.

NAME

crypt – password and file encryption functions

SYNOPSIS

```
cc [flag ...] file ... -lcrypt

char *crypt (key, salt)
char *key, *salt;

void setkey (key)
char *key;

void encrypt (block, flag)
char *block;
int flag;

char *des_crypt (key, salt)
char *key, *salt;

void des_setkey (key)
char *key;

void des_encrypt (block, flag)
char *block;
int flag;

int run_setkey (p, key)
int p[2];
char *key;

int run_crypt (offset, buffer, count, p)
long offset;
char *buffer;
unsigned int count;
int p[2];

int crypt_close(p)
int p[2];
```

DESCRIPTION

des_crypt is the password encryption function. It is based on a one way hashing encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.

key is a user's typed password. **salt** is a two-character string chosen from the set [**a-zA-Z0-9.**]; this string is used to perturb the hashing algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password. The first two characters are the salt itself.

The **des_setkey** and **des_encrypt** entries provide (rather primitive) access to the actual hashing algorithm. The argument of **des_setkey** is a character array of length 64 containing only characters with numerical values of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is set into the machine. This is the key that will be used with the hashing algorithm to encrypt the string **block** with the function **des_encrypt**.

The argument to the **des_encrypt** entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the hashing algorithm using the key set by **des_setkey**. If **edflag** is zero, the argument is encrypted; if non-zero, it is decrypted.

Note that decryption is not provided in the international version of **crypt**(3X). If decryption is attempted with the international version of **des_encrypt**, an error message is printed.

crypt, **setkey**, and **encrypt** are front-end routines that invoke **des_crypt**, **des_setkey**, and **des_encrypt** respectively.

The routines **run_setkey** and **run_crypt** are designed for use by applications that need cryptographic capabilities [such as **ed**(1) and **vi**(1)] that must be compatible with the **crypt**(1) user-level utility. **run_setkey** establishes a two-way pipe connection with **crypt**(1), using **key** as the password argument. **run_crypt** takes a block of characters and transforms the cleartext or ciphertext into their ciphertext or cleartext using **crypt**(1). **offset** is the relative byte position from the beginning of the file that the block of text provided in **block** is coming from. **count** is the number of characters in **block**, and **connection** is an array containing indices to a table of input and output file streams. When encryption is finished, **crypt_close** is used to terminate the connection with **crypt**(1).

run_setkey returns -1 if a connection with **crypt**(1) cannot be established. This will occur on international versions of UNIX where **crypt**(1) is not available. If a null key is passed to **run_setkey**, 0 is returned. Otherwise, 1 is returned. **run_crypt** returns -1 if it cannot write output or read input from the pipe attached to **crypt**. Otherwise it returns 0.

DIAGNOSTICS

In the international version of **crypt**(3X), a flag argument of 1 to **des_encrypt** is not accepted, and an error message is printed.

SEE ALSO

crypt(1), login(1), passwd(1), getpass(3C), passwd(4)

WARNINGS

The return value in **crypt** points to static data that are overwritten by each call.

NAME

curses – terminal screen handling and optimization package

TABLE OF CONTENTS

	Page
SYNOPSIS	1
- compiling information	
- summary of parameters used by *curses* routines	
- alphabetical list of *curses* routines showing their parameters, with a key to the section of ROUTINES in which they are discussed.	
DESCRIPTION	6
- An overview of how *curses* routines should be used	
ROUTINES	8
- Each routine is described in the appropriate section:	
Section 1: Overall Screen Manipulation	9
Section 2: Window and Pad Manipulation	10
Section 3: Output	12
Section 4: Input	16
Section 5: Output Options Setting	19
Section 6: Input Options Setting	20
Section 7: Environment Queries	22
Section 8: Color Manipulation	22
Section 9: Soft Labels	23
Section 10: Low-Level *curses* Access	24
Section 11: Terminfo-Level Manipulations	26
Section 12: Termcap Emulation	29
Section 13: Miscellaneous	29
Section 14: Use of **curscr**	29
Section 15: Obsolete Calls	30
ATTRIBUTES	30
COLORS	30
FUNCTION KEYS	31
LINE GRAPHICS	33

SYNOPSIS

cc [flag ...] file ... **-lcurses** [library ...]

#include <curses.h> (automatically includes **<stdio.h>**, **<termio.h>**, and **<unctrl.h>**).

The parameters in the following summary are the arguments used by the *curses* library routines: they are not global variables. All routines return the **int** values **ERR** or **OK** unless it is stated otherwise in the **ROUTINES** section. Routines that return pointers always return **NULL** on error. (**ERR**, **OK**, and **NULL** are all defined in *<curses.h>*.) The summary lists both the variable definitions and the routine names.

VARIABLE DEFINITIONS

bool bf;
char **area,*boolnames[ ], *boolcodes[ ], *boolfnames[ ], *bp;
char *cap, *capname, codename[2], erasechar, *filename, *fmt;
char *keyname, killchar, *label, *longname;
char *name, *numnames[ ], *numcodes[ ], *numfnames[ ];
char *slk_label, *str, *strnames[ ], *strcodes[ ], *strfnames[ ];
char *term, *tgetstr, *tigetstr, *tgoto, *tparm, *type;
chtype attrs, ch, horch, vertch;
FILE *infd, *outfd;
int begin_x, begin_y, begline, bot, c, col, count;
int dmaxcol, dmaxrow, dmincol, dminrow, *errret, fildes;
int (*init( )), labfmt, labnum, line;
int ms, ncols, new, newcol, newrow, nlines, numlines;
int oldcol, oldrow, overlay;
int p1, p2, p9, pmincol, pminrow, (*putc( )), row;
int smaxcol, smaxrow, smincol, sminrow, start;
int tenths, top, visibility, x, y;
short pair, color, f, r, g, b;
SCREEN *new, *newterm, *set_term;
TERMINAL *cur_term, *nterm, *oterm;
va_list varglist;
WINDOW *curscr, *dstwin, *initscr, *newpad, *newwin, *orig;
WINDOW *pad, *srcwin, *stdscr, *subpad, *subwin, *win;

ROUTINE NAMES

addch(ch)
addstr(str)
attroff(attrs)
attron(attrs)
attrset(attrs)
baudrate( )
beep( )
box(win, vertch, horch)
can_change_color( )
cbreak( )
clear( )
clearok(win, bf)
clrtobot( )
clrtoeol( )
color_content(color, &r, &g, &b)
copywin(srcwin, dstwin, sminrow, smincol,
dminrow, dmincol, dmaxrow, dmaxcol, overlay)
curs_set(visibility)
def_prog_mode( )

ROUTINE NAMES
def_shell_mode()
del_curterm(oterm)
delay_output(ms)
delch()
deleteln()
delwin(win)
doupdate()
draino(ms)
echo()
echochar(ch)
endwin()
erase()
erasechar()
filter()
flash()
flushinp()
garbagedlines(win, begline, numlines)
getbegyx(win, y, x)
getch()
getmaxyx(win, y, x)
getstr(str)
getsyx(y, x)
getyx(win, y, x)
halfdelay(tenths)
has_colors()
has_ic()
has_il()
idlok(win, bf)
inch()
init_color(color, r, g, b)
init_pair(pair, f, b)
initscr()
insch(ch)
insertln()
intrflush(win, bf)
isendwin()
keyname(c)
keypad(win, bf)
killchar()
leaveok(win, bf)
longname()
meta(win, bf)
move(y, x)
mvaddch(y, x, ch)
mvaddstr(y, x, str)
mvcur(oldrow, oldcol, newrow, newcol)

ROUTINE NAMES
mvdelch(y, x)
mvgetch(y, x)
mvgetstr(y, x, str)
mvinch(y, x)
mvinsch(y, x, ch)
mvprintw(y, x, fmt [, arg...])
mvscanw(y, x, fmt [, arg...])
mvwaddch(win, y, x, ch)
mvwaddstr(win, y, x, str)
mvwdelch(win, y, x)
mvwgetch(win, y, x)
mvwgetstr(win, y, x, str)
mvwin(win, y, x)
mvwinch(win, y, x)
mvwinsch(win, y, x, ch)
mvwprintw(win, y, x, fmt [, arg...])
mvwscanw(win, y, x, fmt [, arg...])
napms(ms)
newpad(nlines, ncols)
newterm(type, outfd, infd)
newwin(nlines, ncols, begin_y, begin_x)
nl()
nocbreak()
nodelay(win, bf)
noecho()
nonl()
noraw()
notimeout(win, bf)
overlay(srcwin, dstwin)
overwrite(srcwin, dstwin)
pair_content(pair, &f, &b)
pechochar(pad, ch)
pnoutrefresh(pad, pminrow, pmincol, sminrow,
smincol, smaxrow, smaxcol)
prefresh(pad, pminrow, pmincol, sminrow,
smincol, smaxrow, smaxcol)
printw(fmt [, arg...])
putp(str)
raw()
refresh()
reset_prog_mode()
reset_shell_mode()
resetty()
restartterm(term, fildes, errret)
ripoffline(line, init)
savetty()

ROUTINE NAMES
scanw(fmt [, arg...])
scr_dump(filename)
scr_init(filename)
scr_restore(filename)
scroll(win)
scrollok(win, bf)
set_curterm(nterm)
set_term(new)
setscrreg(top, bot)
setsyx(y, x)
setupterm(term, fildes, errret)
slk_attroff(attrs)
slk_attron(attrs)
slk_attrset(attrs)
slk_clear()
slk_init(fmt)
slk_label(labnum)
slk_noutrefresh()
slk_refresh()
slk_restore()
slk_set(labnum, label, fmt)
slk_touch()
standend()
standout()
start_color()
subpad(orig, nlines, ncols, begin_y, begin_x)
subwin(orig, nlines, ncols, begin_y, begin_x)
tgetent(bp, name)
tgetflag(codename)
tgetnum(codename)
tgetstr(codename, area)
tgoto(cap, col, row)
tigetflag(capname)
tigetnum(capname)
tigetstr(capname)
touchline(win, start, count)
touchwin(win)
tparm(str, p1, p2, ..., p9)
tputs(str, count, putc)
traceoff()
traceon()
typeahead(fildes)
unctrl(c)
ungetch(c)
vidattr(attrs)
vidputs(attrs, putc)

ROUTINE NAMES
vwprintw(win, fmt, varglist)
vwscanw(win, fmt, varglist)
waddch(win, ch)
waddstr(win, str)
wattroff(win, attrs)
wattron(win, attrs)
wattrset(win, attrs)
wclear(win)
wclrtobot(win)
wclrtoeol(win)
wdelch(win)
wdeleteln(win)
wechochar(win, ch)
werase(win)
wgetch(win)
wgetstr(win, str)
winch(win)
winsch(win, ch)
winsertln(win)
wmove(win, y, x)
wnoutrefresh(win)
wprintw(win, fmt [, arg...])
wrefresh(win)
wscanw(win, fmt [, arg...])
wsetscrreg(win, top, bot)
wstandend(win)
wstandout(win)

DESCRIPTION

The *curses* routines give the user a terminal-independent method of updating screens with reasonable optimization.

The file **<curses.h>** must be included at the beginning of programs that use any *curses* routines. In addition, the routine **initscr**( ) or **newterm**( ) must be called before any of the other routines that deal with windows and screens are used. (Three exceptions are noted where they apply.) The routine **endwin**( ) must be called before exiting. To get character-at-a-time input without echoing (most interactive, screen-oriented programs want this), after calling **initscr**( ) you should call **"cbreak**( ); **noecho**( );" Most programs would additionally call **"nonl**( ); **intrflush (stdscr, FALSE); keypad(stdscr, TRUE);".**

Before a *curses* program is run, a terminal's tab stops should be set and its initialization strings, if defined, must be output. To do this, execute **tput init** after the shell environment variable **TERM** has been set and exported. For further details, see *profile*(4), *tput*(1), and the "Tabs and Initialization" subsection of *terminfo*(4).

The *curses* library contains routines that manipulate data structures called *windows* that can be thought of as two-dimensional arrays of characters representing all or part of a terminal screen. A default window called **stdscr** is supplied, which is the size of the terminal screen. Others may be created with **newwin**( ). Windows are referred to by variables declared as **WINDOW** *; the

type **WINDOW** is defined in **<curses.h>** to be a structure. These data structures are manipulated with routines described below, among which the most basic are **move**( ) and **addch**( ). (More general versions of these routines are included, with names beginning with **w**, allowing you to specify a window. The routines not beginning with **w** usually affect **stdscr**.) Then **refresh**( ) is called, telling the routines to make the user's terminal screen look like **stdscr**. The characters in a window are actually of type **chtype**, defined in **<curses.h>**, so that other information about the character may also be stored with each character.

Special windows called *pads* may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be displayed completely. See the description of **newpad**( ) under "Window and Pad Manipulation" for more information.

In addition to drawing characters on the screen, video attributes may be included which cause the characters to be underlined or shown in reverse video on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, *curses* is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in **<curses.h>**, such as **A_REVERSE**, **ACS_HLINE**, and **KEY_LEFT**.

Routines that manipulate color on color alphanumeric terminals are new in this release of *curses*. To use these routines **start_color**( ) must be called, usually right after **initscr**( ). Colors are always used in pairs (referred to as color-pairs). A color-pair consists of a foregound color (for characters) and a background color (for the field the characters are displayed on). A programmer initializes a color-pair with the routine **init_pair**( ). After it has been initialized, **COLOR_PAIR**(n), a macro defined in **<curses.h>**, can be used in the same ways other video attributes can be used. If a terminal is capable of redefining colors the programmer can use the routine **init_color**( ) to change the definition of a color. The routines **has_color**( ) and **can_change_color**( ) return **TRUE** or **FALSE**, depending on whether the terminal has color capabilities and whether the user can change the colors. The routine **color_content**( ) allows a user to identify the amounts of red, green, and blue components in an initialized color. The routine **pair_content**( ) allows a user to find out how a given color-pair is currently defined.

curses also defines the **WINDOW** * variable, **curscr**, which is used only for certain low-level operations like clearing and redrawing a garbaged screen. **curscr** can be used in only a few routines. If the window argument to **clearok**( ) is **curscr**, the next call to **wrefresh**( ) with any window will cause the screen to be cleared and repainted from scratch. If the window argument to **wrefresh**( ) is **curscr**, the screen is immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" function. More information on using **curscr** is provided where its use is appropriate.

The environment variables **LINES** and **COLUMNS** may be set to override **terminfo**'s idea of how large a screen is. These may be used in an AT&T Teletype 5620 layer, for example, where the size of a screen is changeable.

If the environment variable **TERMINFO** is defined, any program using *curses* will check for a local terminal definition before checking in the standard place. For example, if the environment variable **TERM** is set to **att4425**, then the compiled terminal definition is found in */usr/lib/terminfo/a/att4425*. (The **a** is copied from the first letter of **att4425** to avoid creation of huge directories.) However, if **TERMINFO** is set to *$HOME/myterms*, *curses* will first check *$HOME/myterms/a/att4425*, and, if that fails, will then check */usr/lib/terminfo/a/att4425*. This is useful for developing experimental definitions or when write permission on */usr/lib/terminfo* is not available.

The integer variables **LINES** and **COLS** are defined in **<curses.h>**, and will be initialized by **initscr**( ) with the size of the screen. (For more information, see the subsection "Terminfo-Level Manipulations".) The integer variables **COLORS** and **COLOR_PAIRS** are also defined in **<curses.h>** and contain, respectively, the maximum number of colors and color-pairs the terminal can support. They are initialized by **start_color**( ). The constants **TRUE** and **FALSE** have the values **1** and **0**, respectively. The constants **ERR** and **OK** are returned by routines to indicate whether the routine successfully completed. These constants are also defined in **<curses.h>**.

ROUTINES

Many of the following routines have two or more versions. The routines prefixed with **w** require a *window* argument. The routines prefixed with **p** require a *pad* argument. Those without a prefix generally use **stdscr**.

The routines prefixed with **mv** require *y* and *x* coordinates to move to before performing the appropriate action. The **mv**( ) routines imply a call to **move**( ) before the call to the other routine. The window argument is always specified before the coordinates. *y* always refers to the row (of the window), and *x* always refers to the column. The upper left corner is always (**0**,**0**), not (**1**,**1**). The routines prefixed with **mvw** take both a *window* argument and *y* and *x* coordinates.

In each case, *win* is the window affected and *pad* is the pad affected. (**win** and **pad** are always of type **WINDOW** *.) Option-setting routines require a boolean flag *bf* with the value **TRUE** or **FALSE**. (*bf* is always of type **bool**.) The types **WINDOW**, **bool**, and **chtype** are defined in **<curses.h>**. See the SYNOPSIS for a summary of what types all variables are.

All routines return either the integer **ERR** or the integer **OK**, unless otherwise noted. Routines that return pointers always return **NULL** on error.

Sometimes the description of a routine refers to a second routine. If the routine referred to is prefixed with a **w**, then you should assume that other versions of the second routine behave similarly. For example, the description of **initscr**( ) refers to **wrefresh**( ). This implies that the same result will occur if **refresh**( ) is called.

Section 1: Overall Screen Manipulation

WINDOW *initscr()

The first routine called should almost always be **initscr()**. (The exceptions are **slk_init()**, **filter()**, and **ripoffline()**.) This will determine the terminal type and initialize all *curses* data structures. **initscr()** also arranges that the first call to **wrefresh()** will clear the screen. If errors occur, **initscr()** will write an appropriate error message to standard error and exit; otherwise, a pointer to **stdscr** is returned. If the program wants an indication of error conditions, **newterm()** should be used instead of **initscr()**. **initscr()** should only be called once per application.

endwin() A program should always call **endwin()** before exiting or escaping from *curses* mode temporarily, to do a shell escape or *system*(3S) call, for example. This routine will restore *tty*(7) modes, move the cursor to the lower left corner of the screen and reset the terminal into the proper non-visual mode. To resume after a temporary escape, call **wrefresh()** or **doupdate()**.

isendwin()

Returns TRUE if **endwin()** has been called without any subsequent calls to **wrefresh()**.

SCREEN *newterm(type, outfd, infd)

A program that outputs to more than one terminal must use **newterm()** for each terminal instead of **initscr()**. A program that wants an indication of error conditions, so that it may continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, must also use this routine. **newterm()** should be called once for each terminal. It returns a variable of type **SCREEN*** that should be saved as a reference to that terminal. The arguments are the *type* of the terminal to be used in place of the environment variable **TERM**; *outfd*, a *stdio*(3S) file pointer for output to the terminal; and *infd*, another file pointer for input from the terminal. When it is done running, the program must also call **endwin()** for each terminal being used. If **newterm()** is called more than once for the same terminal, the first terminal referred to must be the last one for which **endwin()** is called.

SCREEN *set_term(new)

This routine is used to switch between different terminals. The screen reference *new* becomes the new current terminal. A pointer to the screen of the previous terminal is returned by the routine. This is the only routine which manipulates **SCREEN** pointers; all other routines affect only the current terminal.

Section 2: Window and Pad Manipulation

refresh( )

wrefresh (win)

These routines (or **prefresh**( ), **pnoutrefresh**( ), **wnoutrefresh**( ), or **doupdate**( )) must be called to write output to the terminal, as most other routines merely manipulate data structures. **wrefresh**( ) copies the named window to the physical terminal screen, taking into account what is already there in order to minimize the amount of information that's sent to the terminal (called optimization). **refresh**( ) does the same thing, except it uses **stdscr** as a default window. Unless **leaveok**( ) has been enabled, the physical cursor of the terminal is left at the location of the window's cursor. The number of characters output to the terminal is returned.

Note that **refresh**( ) is a macro.

wnoutrefresh(win)

doupdate( )

These two routines allow multiple updates to the physical terminal screen with more efficiency than **wrefresh**( ) alone. How this is accomplished is described in the next paragraph.

curses keeps two data structures representing the terminal screen: a *physical* terminal screen, describing what is actually on the screen, and a *virtual* terminal screen, describing what the programmer wants to have on the screen. **wrefresh**( ) works by first calling **wnoutrefresh**( ), which copys the named window to the virtual screen, and then by calling **doupdate**( ), which compares the virtual screen to the physical screen and does the actual update. If the programmer wishes to output several windows at once, a series of calls to **wrefresh**( ) will result in alternating calls to **wnoutrefresh**( ) and **doupdate**( ), causing several bursts of output to the screen. By first calling **wnoutrefresh**( ) for each window, it is then possible to call **doupdate**( ) once, resulting in only one burst of output, with probably fewer total characters transmitted and certainly less processor time used.

WINDOW *newwin(nlines, ncols, begin_y, begin_x)

Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The upper left corner of the window is at line *begin_y*, column *begin_x*. If either *nlines* or *ncols* is **0**, they will be set to the value of **lines**–*begin_y* and **cols**–*begin_x*. A new full-screen window is created by calling **newwin(0, 0, 0, 0)**.

mvwin(win, y, x)

Move the window so that the upper left corner will be at position (*y*, *x*). If the move would cause any portion of the window to be moved off the screen, it is an error and the window is not moved.

WINDOW *subwin(orig, nlines, ncols, begin_y, begin_x)

Create and return a pointer to a new window with the given

number of lines (or rows), *nlines*, and columns, *ncols*. The window is at position (*begin_y, begin_x*) on the screen. (This position is relative to the screen, and not to the window *orig*.) The window is made in the middle of the window *orig*, so that changes made to one window will affect the character image of both windows. When changing the image of a subwindow, it will be necessary to call **touchwin**( ) or **touchline**( ) on *orig* before calling **wrefresh**( ) on *orig*.

delwin(win)
Delete the named window, freeing all memory associated with it. If you try to delete a main window before all of its subwindows have been deleted, ERR will be returned.

WINDOW ***newpad**(nlines, ncols)
Create and return a pointer to a new pad data structure with the given number of lines (or rows), *nlines*, and columns, *ncols*. A pad is a window that is not restricted by the screen size and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (e.g. from scrolling or echoing of input) do not occur. It is not legal to call **wrefresh**( ) with a pad as an argument; the routines **prefresh**( ) or **pnoutrefresh**( ) should be called instead.

Note that these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display.

WINDOW ***subpad**(orig, nlines, ncols, begin_y, begin_x)
Create and return a pointer to a subwindow within a pad with the given number of lines (or rows), *nlines*, and columns, *ncols*. Unlike **subwin**( ), which uses screen coordinates, the window is at position (*begin_y, begin_x*) on the pad. The window is made in the middle of the window *orig*, so that changes made to one window will affect the character image of both windows. When changing the image of a subwindow, it will be necessary to call **touchwin**( ) or **touchline**( ) on *orig* before calling **prefresh**( ) on *orig*.

prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
These routines are analogous to **wrefresh**( ) and **wnoutrefresh**( ) except that pads, instead of windows, are involved. The additional parameters are needed to indicate what part of the pad and screen are involved. *pminrow* and *pmincol* specify the upper left corner, in the pad, of the rectangle to be displayed. *sminrow*, *smincol*, *smaxrow*, and *smaxcol* specify the edges, on the screen, of the rectangle to be displayed in. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of *pminrow*, *pmincol*, *sminrow*, or *smincol* are treated as if they were zero.

Section 3: Output

These routines are used to manipulate text in windows.

addch(ch)
waddch(win, ch)
mvaddch(y, x, ch)
mvwaddch(win, y, x, ch)

The character *ch* is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of *putchar* (see *putc*(3S)). At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok**( ) is enabled, the scrolling region will be scrolled up one line.

If *ch* is a tab, newline, or backspace, the cursor will be moved appropriately within the window. A newline also does a **wclrtoeol**( ) before moving. Tabs are considered to be at every eighth column. If *ch* is another control character, it will be drawn in the ^X notation. (Calling **winch**( ) on a position in the window containing a control character will not return the control character, but instead will return one character of the representation of the control character.)

Video attributes can be combined with a character by OR-ing them into the parameter. This will result in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using **winch**( ) and **waddch**( ).) See **wstandout**( ), below.

Note that *ch* is actually of type **chtype**, not a character.

Note that **addch**( ), **mvaddch**( ), and **mvwaddch**( ), are macros.

echochar(ch)
wechochar(win, ch)
pechochar(pad, ch)

These routines are functionally equivalent to a call to **addch**(ch) followed by a call to **refresh**( ), a call to **waddch**(win, ch) followed by a call to **wrefresh**(win), or a call to **waddch**(pad, ch) followed by a call to **prefresh**(pad). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain can be seen by using these routines instead of their equivalents. In the case of **pechochar**( ), the last location of the pad on the screen is reused for the arguments to **prefresh**( ).

Note that *ch* is actually of type **chtype**, not a character.

Note that **echochar**( ) is a macro.

addstr(str)
waddstr(win, str)
mvwaddstr(win, y, x, str)
mvaddstr(y, x, str)

These routines write all the characters of the null-terminated character string *str* on the given window. This is equivalent to calling **waddch**( ) once for each character in the string.

Note that **addstr**( ), **mvaddstr**( ), and **mvwaddstr**( ) are macros.

attroff(attrs)
wattroff(win, attrs)
attron(attrs)
wattron(win, attrs)
attrset(attrs)
wattrset(win, attrs)
standend( )
wstandend(win)
standout( )
wstandout(win)

These routines manipulate the current attributes of the named window. These attributes can be any combination of the constants **A_STANDOUT**, **A_REVERSE**, **A_BOLD**, **A_DIM**, **A_BLINK**, **A_UNDERLINE**, and **A_ALTCHARSET**, as well as the macro **COLOR_PAIR**(n). These attributes are defined in **<curses.h>** and can be combined with the C logical OR (|) operator.

The current attributes of a window are applied to all characters that are written into the window with **waddch**( ). Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations. To the extent possible on the particular terminal, they will be displayed as the graphic rendition of the characters put on the screen.

wattrset(win, attrs) sets the current attributes of the given window to *attrs*. **wattroff**(win, attrs) turns off the named attributes without turning on or off any other attributes. **wattron**(win, attrs) turns on the named attributes without affecting any others. **wstandout**(win, attrs) is the same as **wattron**(win, **A_STANDOUT**). **wstandend**(win, attrs) is the same as **wattrset**(win, **0**), that is, it turns off all attributes.

Note that **wattroff**( ), **wattron**( ), **wattrset**( ), **wstandend**( ), and **wstandout**( ) return **1** at all times.

Note that *attrs* is actually of type **chtype**, not a character.

Note that **attroff**( ), **attron**( ), **attrset**( ), **standend**( ), and **standout**( ) are macros.

beep( )
flash( ) These routines are used to signal the user. **beep**( ) will sound the audible alarm on the terminal, if possible, and if not, will flash the

screen (visible bell), if that is possible. **flash**( ) will flash the screen, and if that is not possible, will sound the audible signal. If neither signal is possible, nothing will happen. Nearly all terminals have an audible signal (bell or beep) but only some can flash the screen.

box(win, vertch, horch)
A box is drawn around the edge of the window, *win*. *vertch* and *horch* are the characters the box is to be drawn with. If *vertch* and *horch* are **0**, then appropriate default characters, **ACS_VLINE** and **ACS_HLINE**, will be used.

Note that *vertch* and *horch* are actually of type **chtype**, not characters.

erase( )
werase(win)
These routines copy blanks to every position in the window.

Note that **erase**( ) is a macro.

clear( )
wclear(win)
These routines are like **erase**( ) and **werase**( ), but they also call **clearok**( ), arranging that the screen will be cleared completely on the next call to **wrefresh**( ) for that window, and repainted from scratch.

Note that **clear**( ) is a macro.

clrtobot( )
wclrtobot(win)
All lines below the cursor in this window are erased. Also, the current line to the right of the cursor, inclusive, is erased.

Note that **clrtobot**( ) is a macro.

clrtoeol( )
wclrtoeol(win)
The current line to the right of the cursor, inclusive, is erased.

Note that **clrtoeol**( ) is a macro.

delay_output(ms)
Insert a *ms* millisecond pause in the output. It is not recommended that this routine be used extensively, because padding characters are used rather than a processor pause.

delch( )
wdelch(win)
mvdelch(y, x)
mvwdelch(win, y, x)
The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change (after moving to (*y*, *x*), if specified). (This does not imply use of the hardware "delete-character" feature.)

Note that **delch**(), **mvdelch**(), and **mvwdelch**() are macros.

deleteln()
wdeleteln(win)
The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change. (This does not imply use of the hardware "delete-line" feature.)

Note that **deleteln**() is a macro.

getyx(win, y, x)
The cursor position of the window is placed in the two integer variables *y* and *x*.

Note that **getyx**() is a macro, so no **"&"** is necessary before the variables *y* and *x*.

getbegyx(win, y, x)
getmaxyx(win, y, x)
The current beginning coordinates (**getbegyx**()) or size (**getmaxyx**()) of the specified window are placed in the two integer variables *y* and *x*.

Note that **getbegyx**() and **getmaxyx**() are macros, so no "&" is necessary before the variables *y* and *x*.

insch(ch)
winsch(win, ch)
mvwinsch(win, y, x, ch)
mvinsch(y, x, ch)
The character *ch* is inserted before the character under the cursor. All characters to the right are moved one space to the right, losing the rightmost character of the line. The cursor position does not change (after moving to (*y*, *x*), if specified). (This does not imply use of the hardware "insert-character" feature.)

Note that *ch* is actually of type **chtype**, not a character.

Note that **insch**(), **mvinsch**(), and **mvwinsch**() are macros.

insertln()
winsertln(win)
A blank line is inserted above the current line and the bottom line is lost. (This does not imply use of the hardware "insert-line" feature.)

Note that **insertln**() is a macro.

move(y, x)
wmove(win, y, x)
The cursor associated with the window is moved to line (row) *y*, column *x*. This does not move the physical cursor of the terminal until **wrefresh**() is called. The position specified is relative to the upper left corner of the window, which is (**0**, **0**).

Note that **move**( ) is a macro.

overlay(srcwin, dstwin)

overwrite(srcwin, dstwin)

These routines overlay text from *srcwin* on top of text from *dstwin* wherever the two windows overlap. The difference is that **overlay**( ) is non-destructive (blanks are not copied), while **overwrite**( ) is destructive.

copywin(srcwin, dstwin, sminrow, smincol, dminrow, dmincol, dmaxrow, dmaxcol, overlay)

This routine provides finer control over the **overlay**( ) and **overwrite**( ) routines. As in the **prefresh**( ) routine, a rectangle is specified in the destination window, (*dminrow*, *dmincol*) and (*dmaxrow*, *dmaxcol*), and the upper-left-corner coordinates of the source window, (*sminrow*, *smincol*). If the argument *overlay* is true, then copying is non-destructive, as in **overlay**( ).

printw(fmt [, arg ...])

wprintw(win, fmt [, arg ...])

mvprintw(y, x, fmt [, arg ...])

mvwprintw(win, y, x, fmt [, arg ...])

These routines are analogous to **printf**(3). The string which would be output by **printf**(3) is instead output using **waddstr**( ) on the given window.

vwprintw(win, fmt, varglist)

This routine corresponds to *vfprintf*(3S). It performs a **wprintw**( ) using a variable argument list. The third argument is a *va_list*, a pointer to a list of arguments, as defined in **<varargs.h>**. See the *vprintf*(3S) and *varargs*(5) manual pages for a detailed description on how to use variable argument lists.

scroll(win)

The window is scrolled up one line. This involves moving the lines in the window data structure.

touchwin(win)

touchline(win, start, count)

Throw away all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change. **touchline**( ) only pretends that *count* lines have been changed, beginning with line *start*.

Section 4: Input

getch( )

wgetch(win)

mvgetch(y, x)

mvwgetch(win, y, x)

A character is read from the terminal associated with the window. In NODELAY mode, if there is no input waiting, the value ERR is returned. In DELAY mode, the program will hang until the system

passes text through to the program. Depending on the setting of **cbreak**( ), this will be after one character (CBREAK mode), or after the first newline (NOCBREAK mode). In HALF-DELAY mode, the program will hang until a character is typed or the specified timeout has been reached. Unless **noecho**( ) has been set, the character will also be echoed into the designated window.

When **wgetch**( ) is called, before getting a character, it will call **wrefresh**( ) if anything in the window has changed (for example, the cursor has moved or text changed).

When using **getch**( ), **wgetch**( ), **mvgetch**( ), or **mvwgetch**( ), do not set both NOCBREAK mode (**nocbreak**( )) and ECHO mode (**echo**( )) at the same time. Depending on the state of the *tty*(7) driver when each character is typed, the program may produce undesirable results.

If **wgetch**( ) encounters a ^**D**, it is returned (unlike *stdio* routines, which would return a null string and have a return code of **-1**).

If **keypad**(win, **TRUE**) has been called, and a function key is pressed, the token for that function key will be returned instead of the raw characters. (See **keypad**( ) under "Input Options Setting.") Possible function keys are defined in **<curses.h>** with integers beginning with **0401**, whose names begin with **KEY_**. If a character is received that could be the beginning of a function key (such as escape), *curses* will set a timer. If the remainder of the sequence is not received within the designated time, the character will be passed through, otherwise the function key value will be returned. For this reason, on many terminals, there will be a delay after a user presses the escape key before the escape is returned to the program. (Use by a programmer of the escape key for a single character routine is discouraged. Also see **notimeout**( ) below.)

Note that **getch**( ), **mvgetch**( ), and **mvwgetch**( ) are macros.

getstr(str)
wgetstr(win, str)
mvgetstr(y, x, str)
mvwgetstr(win, y, x, str)

A series of calls to **wgetch**( ) is made, until a newline, carriage return, or enter key is received. The resulting value (except for this terminating character) is placed in the area pointed at by the character pointer *str*. The user's erase and kill characters are interpreted. See **wgetch**( ) for how it handles characters differently from *stdio* routines (especially ^**D**).

Note that **getstr**( ), **mvgetstr**( ), and **mvwgetstr**( ) are macros.

ungetch(c)

Place *c* onto the input queue, to be returned by the next call to **wgetch**( ).

flushinp()
Throws away any typeahead that has been typed by the user and has not yet been read by the program.

Note that **flushinp**() will not throw away any characters supplied by **ungetch**().

inch()
winch(win)
mvinch(y, x)
mvwinch(win, y, x)
The character, of type **chtype**, at the current position in the named window is returned. If any attributes are set for that position, their values will be OR'ed into the value returned. The predefined constants **A_CHARTEXT** and **A_ATTRIBUTES**, defined in **<curses.h>**, can be used with the C logical AND (&) operator to extract the character or attributes alone.

Note that **inch**(), **winch**(), **mvinch**(), and **mvwinch**() are macros.

scanw(fmt [, arg ...])
wscanw(win, fmt [, arg ...])
mvscanw(y, x, fmt [, arg ...])
mvwscanw(win, y, x, fmt [, arg ...])
These routines correspond to *scanf*(3S), as do their arguments and return values. **wgetstr**() is called on the window, and the resulting line is used as input for the scan. The return value for these routines is the number of *arg* values that are converted by *fmt*. *arg* values that are not converted are lost. See **wgetstr**() for how it handles strings differently than the *stdio* routines (especially ^D).

vwscanw(win, fmt, ap)
This routine is similar to **vwprintw**() in that it performs a **wscanw**() using a variable argument list. The third argument is a *va_list*, a pointer to a list of arguments, as defined in **<varargs.h>**. See the *vprintf*(3S) and *varargs*(5) manual pages for a detailed description on how to use variable argument lists.

Section 5: Output Options Setting

These routines set options within *curses* that deal with output. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling **endwin**( ).

clearok(win, bf)

If enabled (*bf* is **TRUE**), the next call to **wrefresh**( ) with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

idlok(win, bf)

If enabled (*bf* is **TRUE**), *curses* will consider using the hardware "insert/delete-line" feature of terminals so equipped. If disabled (*bf* is FALSE), *curses* will very seldom use this feature. (The "insert/delete-character" feature is always considered.) This option should be enabled only if your application needs "insert/delete-line", for example, for a screen editor. It is disabled by default because "insert/delete-line" tends to be visually annoying when used in applications where it isn't really needed. If "insert/delete-line" cannot be used, *curses* will redraw the changed portions of all lines. Not calling **idlok**( ) saves approximately 5000 bytes of memory.

leaveok(win, bf)

Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

setscrreg(top, bot)

wsetscrreg(win, top, bot)

These routines allow the user to set a software scrolling region in a window. *top* and *bot* are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and **scrollok**( ) are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line. (Note that this has nothing to do with use of a physical scrolling region capability in the terminal, like that in the DEC VT100. Only the text of the window is scrolled; if **idlok**( ) is enabled and the terminal has either a scrolling region or "insert/delete-line" capability, they will probably be used by the output routines.)

Note that **setscrreg**( ) is a macro.

scrollok(win, bf)

This option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled (*bf* is **FALSE**), the cursor is left on the bottom line at the location where the offending character was entered. If

enabled (*bf* is **TRUE**), **wrefresh**( ) is called on the window, and then the physical terminal and window are scrolled up one line. (Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call **idlok**( ).)

Note that **scrollok**( ) will always return OK.

Section 6: Input Options Setting

These routines set options within *curses* that deal with input. The options involve using *ioctl*(2) and therefore interact with *curses* routines. It is not necessary to turn these options off before calling **endwin**( ).

cbreak( )

nocbreak( )

These two routines put the terminal into and out of CBREAK mode, respectively. In CBREAK mode, characters typed by the user are immediately available to the program and erase/kill character processing is not performed. When in NOCBREAK mode, the tty driver will buffer characters typed until a newline or carriage return is typed. Interrupt and flow-control characters are unaffected by this mode (see *termio*(7)). Initially the terminal may or may not be in CBREAK mode, as it is inherited, therefore, a program should call **cbreak**( ) or **nocbreak**( ) explicitly. Most interactive programs using *curses* will set CBREAK mode.

Note that **cbreak**( ) performs a subset of the functionality of **raw**( ). See **wgetch**( ) under "Input" for a discussion of how these routines interact with **echo**( ) and **noecho**( ).

echo( )

noecho( ) These routines control whether characters typed by the user are echoed by **wgetch**( ) as they are typed. Echoing by the tty driver is always disabled, but initially **wgetch**( ) is in ECHO mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling **noecho**( ). See **wgetch**( ) under "Input" for a discussion of how these routines interact with **cbreak**( ) and **nocbreak**( ).

nl( )

nonl( ) These routines control whether carriage return is translated into newline on input by **wgetch**( ). Initially, this translation is done; **nonl**( ) turns the translation off. Note that translation by the *tty*(7) driver is disabled in CBREAK mode.

halfdelay(tenths)

Half-delay mode is similar to CBREAK mode in that characters typed by the user are immediately available to the program. However, after blocking for *tenths* tenths of seconds, ERR will be returned if nothing has been typed. *tenths* must be a number between 1 and 255. Use **nocbreak**( ) to leave half-delay mode.

intrflush(win, bf)

If this option is enabled, when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue

will be flushed, giving the effect of faster response to the interrupt, but causing *curses* to have the wrong idea of what is on the screen. Disabling the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

keypad(win, bf)
This option enables *curses* to obtain information from the keypad of the user's terminal. If enabled, the user can press a function key (such as an arrow key) and **wgetch**( ) will return a single value representing the function key, as in **KEY_LEFT**. If disabled, *curses* will not treat function keys specially and the program would have to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit), calling **keypad** (*win*, **TRUE**) will turn it on.

meta(win, bf)
Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the tty driver (see *termio*(7)). To force 8 bits to be returned, invoke **meta** (*win*, **TRUE**). To force 7 bits to be returned, invoke **meta** (*win*, **FALSE**). The window argument, *win*, is always ignored. If the *terminfo*(4) capabilities **smm** (meta_on) and **rmm** (meta_off) are defined for the terminal, **smm** will be sent to the terminal when **meta** (*win*, **TRUE**) is called and **rmm** will be sent when **meta** (*win*, **FALSE**) is called.

nodelay(win, bf)
This option causes **wgetch**( ) to be a non-blocking call. If no input is ready, **wgetch**( ) will return **ERR**. If disabled, **wgetch**( ) will hang until a key is pressed.

notimeout(win, bf)
While interpreting an input escape sequence, **wgetch**( ) will set a timer while waiting for the next character. If **notimeout**(*win*, **TRUE**) is called, then **wgetch**( ) will not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

raw( )
noraw( ) The terminal is placed into or out of RAW mode. RAW mode is similar to CBREAK mode, in that characters typed are immediately passed through to the user program; however, in RAW mode, the interrupt, quit, suspend, and flow control characters are passed through uninterpreted, instead of generating a signal as they do in CBREAK mode. The behavior of the BREAK key depends on other bits in the *tty*(7) driver that are not set by *curses*.

typeahead(fildes)
curses does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update will be postponed until **wrefresh**( ) or **doupdate**( ) is called again. This allows faster response to commands typed in advance. Normally, the file descriptor for the input FILE pointer passed to **newterm**( ), or **stdin** in the case that **initscr**( ) was used, will be used to do this

typeahead checking. The **typeahead**( ) routine specifies that the file descriptor *fildes* is to be used to check for typeahead instead. If *fildes* is **-1**, then no typeahead checking will be done.

Note that *fildes* is a file descriptor, not a **<stdio.h>** FILE pointer.

Section 7: Environment Queries

baudrate( )
Returns the output speed of the terminal. The number returned is in bits per second, for example, 9600, and is an integer.

char erasechar( )
The user's current erase character is returned.

has_ic( ) True if the terminal has insert- and delete-character capabilities.

has_il( ) True if the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions. This might be used to check to see if it would be appropriate to turn on physical scrolling using **scrollok**( ) or **idlok**( ).

char killchar( )
The user's current line-kill character is returned.

char *longname( )
This routine returns a pointer to a static area containing a verbose description of the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to **initscr**( ) or **newterm**( ). The area is overwritten by each call to **newterm**( ) and is not restored by **set_term**( ), so the value should be saved between calls to **newterm**( ) if **longname**( ) is going to be used with multiple terminals.

Section 8: Color Manipulation

This section describes the color manipulation routines introduced in this release of *curses*.

start_color( )
This routine requires no arguments. It must be called if the user wants to use colors, and before any other color manipulation routine is called. It is good practice to call this routine right after **initscr**( ). **start_color**( ) initializes eight basic colors (black, blue, green, cyan, red, magenta, yellow, and white), and two global variables, **COLORS** and **COLOR_PAIRS** (respectively defining the maximum number of colors and color-pairs the terminal can support). It also restores the terminal's colors to the values they had when the terminal was just turned on.

init_pair(pair, f, b)
This routine changes the definition of a color-pair. It takes three arguments: the number of the color-pair to be changed, the foreground color number, and the background color number. The value of the first argument must be between 1 and **COLOR_PAIRS**-1. The value of the second and third arguments must be between 0 and

COLORS-1. If the color-pair was previously initialized, the screen will be refreshed and all occurrences of that color-pair will be changed to the new definition.

init_color(color, r, g, b)
This routine changes the definition of a color. It takes four arguments: the number of the color to be changed followed by three RGB values (for the amounts of red, green, and blue components). The value of the first argument must be between 0 and **COLORS**-1. (See the section COLOR for the default color index.) The last three arguments must each be a value between 0 and 1000. When **init_color**( ) is used, all occurrences of that color on the screen immediately change to the new definition.

has_colors( )
This routine requires no arguments. It returns **TRUE** if the terminal can manipulate colors, **FALSE** otherwise. This routine facilitates writing terminal-independent programs. For example, a programmer can use it to decide whether to use color or some other video attribute.

can_change_color( )
This routine requires no arguments. It returns **TRUE** if the terminal supports colors and can change their definitions, **FALSE** otherwise. This routine facilitates writing terminal-independent programs.

color_content(color, &r, &g, &b)
This routine gives users a way to find the intensity of the red, green, and blue (RGB) components in a color. It requires four arguments: the color number, and three addresses of **shorts** for storing the information about the amounts of red, green, and blue components in the given color. The value of the first argument must be between 0 and **COLORS**-1. The values that will be stored at the addresses pointed to by the last three arguments will be between 0 (no component) and 1000 (maximum amount of component).

pair_content(pair, &f, &b)
This routine allows users to find out what colors a given color-pair consists of. It requires three arguments: the color-pair number, and two addresses of **shorts** for storing the foreground and the background color numbers. The value of the first argument must be between 1 and **COLOR_PAIRS**-1. The values that will be stored at the addresses pointed to by the second and third arguments will be between 0 and **COLORS**-1.

Section 9: Soft Labels

If desired, *curses* will manipulate the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, *curses* can simulate them by taking over the bottom line of **stdscr**, reducing the size of **stdscr** and the variable **LINES**. *curses* standardizes on 8 labels of 8 characters each. If a *curses* program changes the values of the soft labels, it can restore them only to the default settings for that terminal. (Note also that

soft labels are shown in reverse video by default.) Therefore, if before calling a *curses* program a user changes the values of the soft labels, those values cannot be reset when the *curses* program terminates.

slk_init(labfmt)
: In order to use soft labels, this routine must be called before **initscr**() or **newterm**() is called. If **initscr**() winds up using a line from **stdscr** to emulate the soft labels, then *labfmt* determines how the labels are arranged on the screen. Setting *labfmt* to **0** indicates that the labels are to be arranged in a 3-2-3 arrangement; **1** asks for a 4-4 arrangement.

slk_set(labnum, label, labfmt)
: *labnum* is the label number, from 1 to 8. *label* is the string to be put on the label, up to 8 characters in length. A **NULL** string or a **NULL** pointer will put up a blank label. *labfmt* is one of **0**, **1** or **2**, to indicate whether the label is to be left-justified, centered, or right-justified within the label.

slk_refresh()
slk_noutrefresh()
: These routines correspond to the routines **wrefresh**() and **wnoutrefresh**(). Most applications would use **slk_noutrefresh**() because a **wrefresh**() will most likely soon follow.

char *slk_label(labnum)
: The current label for label number *labnum* is returned, in the same format as it was in when it was passed to **slk_set**(); that is, how it looked prior to being justified according to the *labfmt* argument of **slk_set**().

slk_clear()
: The soft labels are cleared from the screen.

slk_restore()
: The soft labels are restored to the screen after a **slk_clear**().

slk_touch()
: All of the soft labels are forced to be output the next time a **slk_noutrefresh**() is performed.

slk_attron(attrs)
slk_attrset(attrs)
slk_attroff(attrs)
: These routines correspond to **attron**(), **attrset**(), and **attroff**(). They will have effect only if soft labels are simulated on the bottom line of the screen.

Section 10: Low-Level *curses* Access

The following routines give low-level access to various *curses* functionality. These routines typically would be used inside library routines.

def_prog_mode()
def_shell_mode()
: Save the current terminal modes as the "program" (in **curses**) or "shell" (not in **curses**) state for use by the **reset_prog_mode**() and **reset_shell_mode**() routines. This is done automatically by

initscr( ).

reset_prog_mode( )

reset_shell_mode( )

Restore the terminal to "program" (in **curses**) or "shell" (out of *curses*) state. These are done automatically by **endwin**( ) and **doupdate**( ) after an **endwin**( ), so they normally would not be called.

resetty( )

savetty( ) These routines save and restore the state of the terminal modes. **savetty**( ) saves the current state of the terminal in a buffer and **resetty**( ) restores the state to what it was at the last call to **savetty**( ).

getsyx(y, x)

The current coordinates of the virtual screen cursor are returned in *y* and *x*. If **leaveok**( ) is currently **TRUE**, then **–1**,**–1** will be returned. If lines have been removed from the top of the screen using **ripoffline**( ), *y* and *x* include these lines; therefore, *y* and *x* should be used only as arguments for **setsyx**( ).

Note that **getsyx**( ) is a macro, so no "**&**" is necessary before the variables *y* and *x*.

setsyx(y, x)

The virtual screen cursor is set to *y*, *x*. If *y* and *x* are both **–1**, then **leaveok**( ) will be set. The two routines **getsyx**( ) and **setsyx**( ) are designed to be used by a library routine which manipulates *curses* windows but does not want to change the current position of the program's cursor. The library routine would call **getsyx**( ) at the beginning, do its manipulation of its own windows, do a **wnoutrefresh**( ) on its windows, call **setsyx**( ), and then call **doupdate**( ).

ripoffline(line, init)

This routine provides access to the same facility that **slk_init**( ) uses to reduce the size of the screen. **ripoffline**( ) must be called before **initscr**( ) or **newterm**( ) is called. If *line* is positive, a line will be removed from the top of **stdscr**; if negative, a line will be removed from the bottom. When this is done inside **initscr**( ), the routine *init*( ) is called with two arguments: a window pointer to the 1-line window that has been allocated and an integer with the number of columns in the window. Inside this initialization routine, the integer variables **LINES** and **COLS** (defined in **<curses.h>**) are not guaranteed to be accurate and **wrefresh**( ) or **doupdate**( ) must not be called. It is allowable to call **wnoutrefresh**( ) during the initialization routine.

ripoffline( ) can be called up to five times before calling **initscr**( ) or **newterm**( ).

scr_dump(filename)

The current contents of the virtual screen are written to the file *filename*.

scr_restore(filename)
The virtual screen is set to the contents of *filename*, which must have been written using **scr_dump**( ). **ERR** is returned if the contents of *filename* are not compatible with the current release of *curses* software. The next call to **doupdate**( ) will restore the screen to what it looked like in the dump file.

scr_init(filename)
The contents of *filename* are read in and used to initialize the *curses* data structures about what the terminal currently has on its screen. If the data is determined to be valid, *curses* will base its next update of the screen on this information rather than clearing the screen and starting from scratch. **scr_init**( ) would be used after **initscr**( ) or a *system*(3S) call to share the screen with another process which has done a **scr_dump**( ) after its **endwin**( ) call. The data will be declared invalid if the *terminfo*(4) capability **nrrmc** is true or the time-stamp of the tty is old. Note that **keypad**( ), **meta**( ), **slk_clear**( ), **curs_set**( ), **flash**( ), and **beep**( ) do not affect the contents of the screen, but will make the tty's time-stamp old.

curs_set(visibility)
The cursor state is set to invisible, normal, or very visible for *visibility* equal to **0**, **1** or **2**. If the terminal supports the *visibility* requested, the previous *cursor* state is returned; otherwise, **ERR** is returned.

draino(ms)
Wait until the output has drained enough that it will only take *ms* more milliseconds to drain completely.

garbagedlines(win, begline, numlines)
This routine indicates to *curses* that a screen line is garbaged and should be thrown away before having anything written over the top of it. It could be used for programs such as editors which want a command to redraw just a single line. Such a command could be used in cases where there is a noisy communications line and redrawing the entire screen would be subject to even more communication noise. Just redrawing the single line gives some semblance of hope that it would show up unblemished. The current location of the window is used to determine which lines are to be redrawn.

napms(ms)
Sleep for *ms* milliseconds.

mvcur(oldrow, oldcol, newrow, newcol)
Low-level cursor motion.

Section 11: Terminfo-Level Manipulations

These low-level routines must be called by programs that need to deal directly with the *terminfo*(4) database to handle certain terminal capabilities, such as programming function keys. For all other functionality, *curses* routines are more suitable and their use is recommended.

Initially, **setupterm**( ) should be called. (Note that **setupterm**( ) is automatically called by **initscr**( ) and **newterm**( ).) This will define the set of terminal-dependent variables defined in the *terminfo*(4) database. The *terminfo*(4) variables **lines** and **columns** (see *terminfo*(4)) are initialized by **setupterm**( ) as follows: if the environment variables **LINES** and **COLUMNS** exist, their values are used. If the above environment variables do not exist and the program is running in a layer (see *layers*(1)), the size of the current layer is used. Otherwise, the values for **lines** and **columns** specified in the *terminfo*(4) database are used.

The header files **<curses.h>** and **<term.h>** should be included, in this order, to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through **tparm**( ) to instantiate them. All *terminfo*(4) strings (including the output of **tparm**( )) should be printed with **tputs**( ) or **putp**( ). Before exiting, **reset_shell_mode**( ) should be called to restore the tty modes. Programs which use cursor addressing should output **enter_ca_mode** upon startup and should output **exit_ca_mode** before exiting (see *terminfo*(4)). (Programs desiring shell escapes should call **reset_shell_mode**( ) and output **exit_ca_mode** before the shell is called and should output **enter_ca_mode** and call **reset_prog_mode**( ) after returning from the shell. Note that this is different from the *curses* routines (see **endwin**( )).

setupterm(term, fildes, errret)

Reads in the *terminfo*(4) database, initializing the *terminfo*(4) structures, but does not set up the output virtualization structures used by *curses*. The terminal type is in the character string *term*: if *term* is **NULL**, the environment variable **TERM** will be used. All output is to the file descriptor *fildes*. If *errret* is not **NULL**, then **setupterm**( ) will return **OK** or **ERR** and store a status value in the integer pointed to by *errret*. A status of **1** in *errret* is normal, **0** means that the terminal could not be found, and **–1** means that the *terminfo*(4) database could not be found. If *errret* is **NULL**, **setupterm**( ) will print an error message upon finding an error and exit. Thus, the simplest call is **setupterm ((char *)0, 1, (int *)0)**, which uses all the defaults.

The *terminfo*(4) boolean, numeric and string variables are stored in a structure of type **TERMINAL**. After **setupterm**( ) returns successfully, the variable **cur_term** (of type **TERMINAL ***) is initialized with all of the information that the *terminfo*(4) boolean, numeric and string variables refer to. The pointer may be saved before calling **setupterm**( ) again. Further calls to **setupterm**( ) will allocate new space rather than reuse the space pointed to by **cur_term**.

set_curterm(nterm)

nterm is of type **TERMINAL ***. **set_curterm**( ) sets the variable **cur_term** to *nterm*, and makes all of the *terminfo*(4) boolean, numeric and string variables use the values from *nterm*.

del_curterm(oterm)

oterm is of type **TERMINAL ***. **del_curterm**( ) frees the space

pointed to by *oterm* and makes it available for further use. If *oterm* is the same as **cur_term**, then references to any of the *terminfo*(4) boolean, numeric and string variables thereafter may refer to invalid memory locations until another **setupterm**( ) has been called.

restartterm(term, fildes, errret)
Similar to **setupterm**( ), except that it is called after restoring memory to a previous state; for example, after a call to **scr_restore**( ). It assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different.

char *tparm(str, p1, p2, ..., p9)
Instantiate the string *str* with parms p*i*. A pointer is returned to the result of *str* with the parameters applied.

tputs(str, count, putc)
Apply padding to the string *str* and output it. *str* must be a *terminfo*(4) string variable or the return value from **tparm**( ), **tgetstr**( ), **tigetstr**( ) or **tgoto**( ). *count* is the number of lines affected, or **1** if not applicable. *putc* is a *putchar*(3S)-like routine to which the characters are passed, one at a time.

putp(str) A routine that calls **tputs** (*str*, **1**, **putchar**).

vidputs(attrs, putc)
Output a string that puts the terminal in the video attribute mode *attrs*, which is any combination of the attributes listed below. The characters are passed to the *putchar*(3S)-like routine *putc*( ).

vidattr(attrs)
Similar to **vidputs**( ), except that it outputs through *putchar*(3S).

The following routines return the value of the capability corresponding to the character string containing the *terminfo*(4) *capname* passed to them. For example, **rc = tigetstr("acsc")** causes the value of **acsc** to be returned in **rc**.

tigetflag(capname)
The value **–1** is returned if *capname* is not a boolean capability.
The value **0** is returned if *capname* is not defined for this terminal.

tigetnum(capname)
The value **–2** is returned if *capname* is not a numeric capability.
The value **–1** is returned if *capname* is not defined for this terminal.

tigetstr(capname)
The value (char *) **–1** is returned if *capname* is not a string capability. A null value is returned if *capname* is not defined for this terminal.

char *boolnames[ ], ***boolcodes**[ ], ***boolfnames**[ ]
char *numnames[ ], ***numcodes**[ ], ***numfnames**[ ]
char *strnames[ ], ***strcodes**[ ], ***strfnames**[ ]
These null-terminated arrays contain the *capnames*, the *termcap* codes, and the full C names, for each of the *terminfo*(4) variables.

Section 12: Termcap Emulation

These routines are included as a conversion aid for programs that use the *termcap* library. Their parameters are the same and the routines are emulated using the *terminfo*(4) database.

tgetent(bp, name)
Look up *termcap* entry for *name*. The emulation ignores the buffer pointer *bp*.

tgetflag(codename)
Get the boolean entry for *codename*.

tgetnum(codename)
Get numeric entry for *codename*.

char *tgetstr(codename, area)
Return the string entry for *codename*. If *area* is not **NULL**, then also store it in the buffer pointed to by *area* and advance *area*. **tputs**( ) should be used to output the returned string.

char *tgoto(cap, col, row)
Instantiate the parameters into the given capability. The output from this routine is to be passed to **tputs**( ).

tputs(str, affcnt, putc)
See **tputs**( ) above, under "TERMINFO-LEVEL MANIPULATIONS".

Section 13: Miscellaneous

traceoff( )
traceon( )
Turn off and on debugging trace output when using the debug version of the *curses* library. This facility is available only to customers with a source license.

unctrl(c) This macro expands to a character string which is a printable representation of the character *c*. Control characters are displayed in the ^X notation. Printing characters are displayed as is.

unctrl( ) is a macro, defined in **<unctrl.h>**, which is automatically included by **<curses.h>**.

char *keyname(c)
A character string corresponding to the key *c* is returned.

filter( ) This routine is one of the few that is to be called before **initscr**( ) or **newterm**( ) is called. It arranges things so that *curses* thinks that there is a 1-line screen. *curses* will not use any terminal capabilities that assume that they know what line on the screen the cursor is on.

Section 14: Use of curscr

The special window **curscr** can be used in only a few routines. If the window argument to **clearok**( ) is **curscr**, the next call to **wrefresh**( ) with any window will cause the screen to be cleared and repainted from scratch. If the

window argument to **wrefresh**( ) is **curscr**, the screen is immediately cleared and repainted from scratch. (This is how most programs would implement a "repaint-screen" routine.) The source window argument to **overlay**( ), **overwrite**( ), and **copywin**( ) may be **curscr**, in which case the current contents of the virtual terminal screen will be accessed.

Section 15: Obsolete Calls

Various routines are provided to maintain compatibility in programs written for older versions of the curses library. These routines are all emulated as indicated below.

crmode( )	Replaced by **cbreak**( ).
fixterm( )	Replaced by **reset_prog_mode**( ).
gettmode( )	A no-op.
nocrmode( )	Replaced by **nocbreak**( ).
resetterm( )	Replaced by **reset_shell_mode**( ).
saveterm( )	Replaced by **def_prog_mode**( ).
setterm( )	Replaced by **setupterm**( ).

ATTRIBUTES

The following video attributes, defined in **<curses.h>**, can be passed to the routines **wattron**( ), **wattroff**( ), and **wattrset**( ), or OR'ed with the characters passed to **waddch**( ).

A_STANDOUT	Terminal's best highlighting mode
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_ALTCHARSET	Alternate character set
A_NORMAL	Turn all attributes off, for example: **wattrset** (win, **A_NORMAL**)
COLOR_PAIR(n)	Color-pair defined in n (note that this is a macro)

The following bit-masks may be AND'ed with characters returned by **winch**( ).

A_CHARTEXT	Extract character
A_ATTRIBUTES	Extract attributes
A_COLOR	Extract color-pair field information

The following macro is the reverse of COLOR_PAIR(n).

PAIR_NUMBER(attrs)	Returns the pair number associated with the COLOR_PAIR(n) attribute (note that this is a macro)

COLORS

In **<curses.h>** the following macros are defined to have the numeric value shown. These are the default colors. *curses* also assumes that color 0 (zero) is the default background color for all terminals.

COLOR_BLACK	0

COLOR_BLUE	1
COLOR_GREEN	2
COLOR_CYAN	3
COLOR_RED	4
COLOR_MAGENTA	5
COLOR_YELLOW	6
COLOR_WHITE	7

FUNCTION KEYS

The following function keys, defined in **<curses.h>**, might be returned by **wgetch**( ) if **keypad**( ) has been enabled. Note that not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or the definition for the key is not present in the *terminfo*(4) database.

Name	*Value*	*Key name*
KEY_BREAK	0401	break key (unreliable)
KEY_DOWN	0402	The four arrow keys ...
KEY_UP	0403	
KEY_LEFT	0404	
KEY_RIGHT	0405	...
KEY_HOME	0406	Home key (upward+left arrow)
KEY_BACKSPACE	0407	backspace (unreliable)
KEY_F0	0410	Function keys - space for 64 keys reserved.
KEY_F(n)	(KEY_F0+(n))	Formula for f_n.
KEY_DL	0510	Delete line
KEY_IL	0511	Insert line
KEY_DC	0512	Delete character
KEY_IC	0513	Insert char or enter insert mode
KEY_EIC	0514	Exit insert char mode
KEY_CLEAR	0515	Clear screen
KEY_EOS	0516	Clear to end of screen
KEY_EOL	0517	Clear to end of line
KEY_SF	0520	Scroll 1 line forward
KEY_SR	0521	Scroll 1 line backwards (reverse)
KEY_NPAGE	0522	Next page
KEY_PPAGE	0523	Previous page
KEY_STAB	0524	Set tab
KEY_CTAB	0525	Clear tab
KEY_CATAB	0526	Clear all tabs
KEY_ENTER	0527	Enter or send
KEY_SRESET	0530	soft (partial) reset
KEY_RESET	0531	reset or hard reset
KEY_PRINT	0532	print or copy

KEY_LL	0533	home down or bottom (lower left) keypad is arranged like this: A1 up A3 left B2 right C1 down C3
KEY_A1	0534	Upper left of keypad
KEY_A3	0535	Upper right of keypad
KEY_B2	0536	Center of keypad
KEY_C1	0537	Lower left of keypad
KEY_C3	0540	Lower right of keypad
KEY_BTAB	0541	Back tab key
KEY_BEG	0542	beg(inning) key
KEY_CANCEL	0543	cancel key
KEY_CLOSE	0544	close key
KEY_COMMAND	0545	cmd (command) key
KEY_COPY	0546	copy key
KEY_CREATE	0547	create key
KEY_END	0550	end key
KEY_EXIT	0551	exit key
KEY_FIND	0552	find key
KEY_HELP	0553	help key
KEY_MARK	0554	mark key
KEY_MESSAGE	0555	message key
KEY_MOVE	0556	move key
KEY_NEXT	0557	next object key
KEY_OPEN	0560	open key
KEY_OPTIONS	0561	options key
KEY_PREVIOUS	0562	previous object key
KEY_REDO	0563	redo key
KEY_REFERENCE	0564	ref(erence) key
KEY_REFRESH	0565	refresh key
KEY_REPLACE	0566	replace key
KEY_RESTART	0567	restart key
KEY_RESUME	0570	resume key
KEY_SAVE	0571	save key
KEY_SBEG	0572	shifted beginning key
KEY_SCANCEL	0573	shifted cancel key
KEY_SCOMMAND	0574	shifted command key
KEY_SCOPY	0575	shifted copy key
KEY_SCREATE	0576	shifted create key
KEY_SDC	0577	shifted delete char key
KEY_SDL	0600	shifted delete line key
KEY_SELECT	0601	select key
KEY_SEND	0602	shifted end key
KEY_SEOL	0603	shifted clear line key
KEY_SEXIT	0604	shifted exit key
KEY_SFIND	0605	shifted find key

KEY_SHELP	0606	shifted help key
KEY_SHOME	0607	shifted home key
KEY_SIC	0610	shifted input key
KEY_SLEFT	0611	shifted left arrow key
KEY_SMESSAGE	0612	shifted message key
KEY_SMOVE	0613	shifted move key
KEY_SNEXT	0614	shifted next key
KEY_SOPTIONS	0615	shifted options key
KEY_SPREVIOUS	0616	shifted prev key
KEY_SPRINT	0617	shifted print key
KEY_SREDO	0620	shifted redo key
KEY_SREPLACE	0621	shifted replace key
KEY_SRIGHT	0622	shifted right arrow
KEY_SRSUME	0623	shifted resume key
KEY_SSAVE	0624	shifted save key
KEY_SSUSPEND	0625	shifted suspend key
KEY_SUNDO	0626	shifted undo key
KEY_SUSPEND	0627	suspend key
KEY_UNDO	0630	undo key

LINE GRAPHICS

The following variables may be used to add line-drawing characters to the screen with **waddch**(). When defined for the terminal, the variable will have the **A_ALTCHARSET** bit turned on. Otherwise, the default character listed below will be stored in the variable. The names were chosen to be consistent with the DEC VT100 nomenclature.

Name	*Default*	*Glyph Description*
ACS_ULCORNER	+	upper left corner
ACS_LLCORNER	+	lower left corner
ACS_URCORNER	+	upper right corner
ACS_LRCORNER	+	lower right corner
ACS_RTEE	+	right tee
ACS_LTEE	+	left tee
ACS_BTEE	+	bottom tee
ACS_TTEE	+	top tee
ACS_HLINE	–	horizontal line
ACS_VLINE	|	vertical line
ACS_PLUS	+	plus
ACS_S1	–	scan line 1
ACS_S9	_	scan line 9
ACS_DIAMOND	+	diamond
ACS_CKBOARD	:	checker board (stipple)
ACS_DEGREE	'	degree symbol
ACS_PLMINUS	#	plus/minus
ACS_BULLET	o	bullet
ACS_LARROW	<	arrow pointing left

ACS_RARROW	>	arrow pointing right
ACS_DARROW	v	arrow pointing down
ACS_UARROW	^	arrow pointing up
ACS_BOARD	#	board of squares
ACS_LANTERN	#	lantern symbol
ACS_BLOCK	#	solid square block

DIAGNOSTICS

All routines return the integer **OK** upon successful completion and the integer **ERR** upon failure, unless otherwise noted in the preceding routine descriptions.

All macros return the value of their **w** version, except **getsyx**( ), **getyx**( ), **getbegyx**( ), **getmaxyx**( ). For these macros, no useful value is returned.

Routines that return pointers always return **(type *) NULL** on error.

BUGS

Currently typeahead checking is done using a nodelay read followed by an **ungetch**( ) of any character that may have been read. Typeahead checking is done only if **wgetch**( ) has been called at least once. This may change when proper kernel support is available. Programs which use a mixture of their own input routines with *curses* input routines may wish to call **typeahead(-1)** to turn off typeahead checking.

The argument to **napms**( ) is currently rounded up to the nearest second.

draino (ms) only works for *ms* equal to **0**.

WARNINGS

If the new *curses* features are used, code cannot be compiled and linked on systems having UNIX System V Release 3.1 or earliler. Code written for previous releases of *curses* is compatible with this version.

The plotting library *plot*(3X) and the curses library *curses*(3X) both use the names **erase**( ) and **move**( ). The *curses* versions are macros. If you need both libraries, put the *plot*(3X) code in a different source file than the *curses*(3X) code, and/or **#undef move**( ) and **erase**( ) in the *plot*(3X) code.

Between the time a call to **initscr**( ) and **endwin**( ) has been issued, use only the routines in the *curses* library to generate output. Using system calls or the "standard I/O package" (see *stdio*(3S)) for output during that time can cause unpredictable results.

If a pointer passed to a routine as a window argument is null or out of range, the results are undefined (core may be dumped).

SEE ALSO

cc(1), ld(1), tput(1) ioctl(2); plot(3X), putc(3S), scanf(3S), stdio(3S), system(3S), vprintf(3S) profile(4), term(4), terminfo(4), varargs(5), termio(7), tty(7)

NAME

directory: opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir (filename)
char *filename;

struct dirent *readdir (dirp)
DIR *dirp;

long telldir (dirp)
DIR *dirp;

void seekdir (dirp, loc)
DIR *dirp;
long loc;

void rewinddir (dirp)
DIR *dirp;

void closedir(dirp)
DIR *dirp;
```

DESCRIPTION

opendir opens the directory named by **filename** and associates a *directory stream* with it. **opendir** returns a pointer to be used to identify the directory stream in subsequent operations. The file descriptor associated with this directory stream is marked to be closed if the calling process makes a system call belonging to the **exec**(2) family. The pointer **NULL** is returned if **filename** cannot be accessed or is not a directory, or if it cannot **malloc**(3X) enough memory to hold a DIR structure or a buffer for the directory entries.

readdir returns a pointer to the next active directory entry. No inactive entries are returned. It returns **NULL** upon reaching the end of the directory or upon detecting an invalid location in the directory.

telldir returns the current location associated with the named directory stream.

seekdir sets the position of the next **readdir** operation on the directory stream. The new position reverts to the one associated with the directory stream when the **telldir** operation from which **loc** was obtained was performed.

rewinddir resets the position of the named directory stream to the beginning of the directory.

closedir closes the named directory stream and frees the DIR structure.

The following errors can occur as a result of these operations.

opendir:

[ENOTDIR] A component of **filename** is not a directory.

[EACCES] A component of **filename** denies search permission.

[EMFILE] The maximum number of file descriptors are currently open.

[EFAULT] **filename** points outside the allocated address space.

readdir:

[ENOENT] The current file pointer for the directory is not located at a valid entry.

[EBADF] The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

telldir, seekdir, and **closedir:**

[EBADF] The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

EXAMPLES

Sample code which searches a directory for entry **name**:

```
dirp = opendir( "." );
while ( (dp = readdir( dirp )) != NULL )
      if ( strcmp( dp->d_name, name ) == 0 )
            {
            closedir( dirp );
            return FOUND;
            }
closedir( dirp );
return NOT_FOUND;
```

NOTES

When scanning a directory, entries may be missed if they are created after the directory scan has begun. This happens if a created entry is inserted before the current location of the directory pointer. The same thing happens if **telldir** is used to save the directory position. If a directory entry is entered before this location, it is missed when the directory scan continues.

SEE ALSO

getdents(2), dirent(4)

WARNINGS

rewinddir is implemented as a macro, so its function address cannot be taken.

NAME

getspent, getspnam, setspent, endspent, fgetspent, lckpwdf, ulckpwdf – get shadow password file entry

SYNOPSIS

```
#include <shadow.h>

struct spwd *getspent ( )

struct spwd *getspnam (name)
char *name;

int lckpwdf ( )

int ulckpwdf ( )

void setspent ( )

void endspent ( )

struct spwd *fgetspent (fp)
FILE *fp;
```

DESCRIPTION

The **getspent** and **getspnam** routines each return a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/shadow* file. Each line in the file contains a shadow password structure, declared in the **< shadow.h>** header file:

```
struct spwd{
        char   *sp_namp;
        char   *sp_pwdp;
        long   sp_lstchg;
        long   sp_min;
        long   sp_max;
};
```

When first called, **getspent** returns a pointer to the first *spwd* structure in the file; thereafter, it returns a pointer to the next *spwd* structure in the file so successive calls can be used to search the entire file. The **getspnam** routine searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found. The **getspent** and **getspnam** routines populate the **sp_min**, **sp_max**, or **sp_lstchg** field with –1 if the corresponding field in */etc/shadow* is empty. If an end-of-file or an error is encountered on reading, or there is a format error in the file, these functions return a **NULL** pointer.

/etc/.pwd.lock is the lock file. It is used to coordinate modification access to the password files */etc/passwd* and */etc/shadow*. **lckpwdf**(3x) and **ulckpwdf**(3x) are routines that are used to gain modification access to the password files through the lock file. A process first uses **lckpwdf** to lock the lock file thereby gaining exclusive rights to modify the */etc/passwd* or */etc/shadow* password file. Upon completing modifications, a process should release the lock on the lock file via **ulckpwdf**. This prevents simultaneous modification of the password files.

The **lckpwdf** routine attempts to lock the file */etc/.pwd.lock*. If file */etc/.pwd.lock* is already locked, **lckpwdf** tries for 15 seconds to lock the file. If **lckpwdf** is unsuccessful, it returns a –1. If **lckpwdf** succeeds in locking

/etc/.pwd.lock within 15 seconds, then a return code other than –1 is returned.

The **ulckpwdf** routine attempts to unlock the file */etc/.pwd.lock*. If successful, **ulckpwdf returns a 0.** If the unlocking failed, for example in the case where */etc/.pwd.lock* was not locked initially, **ulckpwdf returns a –1.**

A call to **setspent** has the effect of rewinding the shadow password file to allow repeated searches. The **endspent** routine may be called to close the shadow password file when processing is complete.

The **fgetspent** routine returns a pointer to the next *spwd* structure in the stream *fp*, which matches the format of **/etc/shadow.**

FILES

/etc/shadow, /etc/passwd, /etc/.pwd.lock

SEE ALSO

mkpwdbm(1M), passmgmt(1M), putspent(3X)

DIAGNOSTICS

A NULL pointer is returned on EOF or error.

WARNINGS

If a program not otherwise using standard I/O uses this routine, the size of the program increase more than might be expected.

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

ldahread – read the archive header of a member of an archive file

SYNOPSIS

```
#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;
```

DESCRIPTION

If **TYPE(ldptr)** is the archive file magic number, **ldahread** reads the archive header of the common object file currently associated with **ldptr** into the area of memory beginning at **arhead.**

ldahread returns SUCCESS or FAILURE. **ldahread** fails if **TYPE(ldptr)** does not represent an archive file, or if it cannot read the archive header.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose (3X), ldopen(3X), intro(4), ldfcn(4), path.h(4), ldclose(3X), ldopen(3X), ldfcn(4), ar(4)

NAME

ldclose, ldaclose – close a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldclose (ldptr)
LDFILE *ldptr;

int ldaclose (ldptr)
LDFILE *ldptr;
```

DESCRIPTION

ldopen(3X) and **ldclose** are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If **TYPE(ldptr)** does not represent an archive file, **ldclose** closes the file and frees the memory allocated to the **LDFILE** structure associated with **ldptr**.

If **TYPE(ldptr)** is the magic number of an archive file, and if there are any more files in the archive, **ldclose** reinitializes **OFFSET(ldptr)** to the file address of the next archive member and returns **FAILURE**. The **LDFILE** structure is prepared for a subsequent **ldopen**(3X). In all other cases, **ldclose** returns **SUCCESS**.

ldaclose closes the file and frees the memory allocated to the **LDFILE** structure associated with **ldptr** regardless of the value of **TYPE(ldptr)**. **ldaclose** always returns **SUCCESS**. The function is often used in conjunction with **ldaopen**.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

fclose(3S), ldopen(3X), intro(4), ldfcn(4), paths.h(4), fclose(3S), ldopen(3X), ldfcn(4)

NAME

ldfhread – read the file header of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;
```

DESCRIPTION

ldfhread reads the file header of the common object file currently associated with **ldptr** into the area of memory beginning at **filehead.**

ldfhread returns **SUCCESS** or **FAILURE.** **ldfhread** fails if it cannot read the file header.

In most cases the use of **ldfhread** can be avoided by using the macro **HEADER(ldptr)** defined in **ldfcn.h** (see **ldfcn**(4)). **fieldname** of the file header may be accessed using `HEADER(ldptr).fieldname`.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4)

NAME

ldgetname – retrieve symbol name for common object file symbol table entry

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

char *ldgetname (ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;
```

DESCRIPTION

ldgetname returns a pointer to the name associated with **symbol** as a string. The string is contained in a static buffer local to **ldgetname** that is overwritten by each call to **ldgetname**, and therefore must be copied by the caller if the name is to be saved.

ldgetname can be used to retrieve names from object files without any backward compatibility problems. **ldgetname** will return **NULL** (defined in *<stdio.h>*) for an object file if the name cannot be retrieved. This situation can occur in the following cases:

- if the "string table" cannot be found,
- if not enough memory can be allocated for the string table,
- if the string table appears not to be a string table (for example, if an auxiliary entry is handed to **ldgetname** that looks like a reference to a name in a nonexistent string table), or
- if the name's offset into the string table is past the end of the string table.

Typically, **ldgetname** is called immediately after a successful call to **ldtbread** to retrieve the name associated with the symbol table entry filled by **ldtbread**.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbread(3X), ldtbseek(3X), ldfcn(4)

NAME

ldlread, ldlinit, ldlitem – manipulate line number entries of a common object file function

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>

int ldlread(ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO *linent;

int ldlinit(ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO *linent;
```

DESCRIPTION

ldlread searches the line number entries of the common object file currently associated with **ldptr**. **ldlread** begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by **fcnindx**, the index of its entry in the object file symbol table. **ldlread** reads the entry with the smallest line number equal to or greater than **linenum** into the memory beginning at **linent**.

ldlinit and **ldlitem** together perform exactly the same function as **ldlread**. After an initial call to **ldlread** or **ldlinit**, **ldlitem** may be used to retrieve a series of line number entries associated with a single function. **ldlinit** simply locates the line number entries for the function identified by **fcnindx**. **ldlitem** finds and reads the entry with the smallest line number equal to or greater than **linenum** into the memory beginning at **linent**.

ldlread, **ldlinit**, and **ldlitem** each return either SUCCESS or FAILURE. **ldlread** fails if there are no line number entries in the object file, if **fcnindx** does not index a function entry in the symbol table, or if it finds no line number equal to or greater than **linenum**. **ldlinit** fails if there are no line number entries in the object file or if **fcnindx** does not index a function entry in the symbol table. **ldlitem** fails if it finds no line number equal to or greater than **linenum**.

The programs must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbindex(3X), ldfcn(4)

NAME

ldlseek, ldnlseek – seek to line number entries of a section of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldlseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

ldlseek seeks to the line number entries of the section specified by **sectindx** of the common object file currently associated with **ldptr.**

ldnlseek seeks to the line number entries of the section specified by **sectname.**

ldlseek and **ldnlseek** return SUCCESS or **FAILURE.** **ldlseek** fails if **sectindx** is greater than the number of sections in the object file; **ldnlseek** fails if there is no section name corresponding with ***sectname.** Either function fails if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of **one.**

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4)

NAME

ldohseek – seek to the optional file header of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldohseek (ldptr)
LDFILE *ldptr;
```

DESCRIPTION

ldohseek seeks to the optional file header of the common object file currently associated with **ldptr**.

ldohseek returns **SUCCESS** or **FAILURE**. **ldohseek** fails if the object file has no optional header or if it cannot seek to the optional header.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldfhread(3X), ldfcn(4)

NAME

ldopen, ldaopen – open a common object file for reading

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;
```

DESCRIPTION

ldopen and **ldclose**(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If **ldptr** has the value **NULL**, then **ldopen** opens **filename** and allocates and initializes the **LDFILE** structure, and returns a pointer to the structure to the calling program.

If **ldptr** is valid and if **TYPE(ldptr)** is the archive magic number, **ldopen** reinitializes the **LDFILE** structure for the next archive member of **filename**.

ldopen and **ldclose**(3X) are designed to work in concert. **ldclose** returns **FAILURE** only when **TYPE(ldptr)** is the archive magic number and there is another file in the archive to be processed. Only then should **ldopen** be called with the current value of **ldptr**. In all other cases, in particular whenever a new **filename** is opened, **ldopen** should be called with a **NULL ldptr** argument.

The following is a prototype for the use of **ldopen** and **ldclose**(3X).

```
/* for each filename to be processed */
ldptr = NULL;
do
{
        if ( (ldptr = ldopen(filename, ldptr)) != NULL )
        {
                /* check magic number */
                /* process the file */
        }
} while (ldclose(ldptr) == FAILURE );
```

If the value of **oldptr** is not **NULL**, **ldaopen** opens **filename** anew and allocates and initializes a new **LDFILE** structure, copying the **TYPE**, **OFFSET**, and **HEADER** fields from **oldptr**. **ldaopen** returns a pointer to the new **LDFILE** structure. This new pointer is independent of the old pointer, **oldptr**. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

Both **ldopen** and **ldaopen** open **filename** for reading. Both functions return **NULL** if **filename** cannot be opened, or if memory for the **LDFILE** structure cannot be allocated. A successful open does not insure that the given file is a common object file or an archived object file.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

fopen(3S), ldclose(3X), ldfcn(4)

NAME

ldrseek, ldnrseek – seek to relocation entries of a section of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

ldrseek seeks to the relocation entries of the section specified by **sectindx** of the common object file currently associated with **ldptr.**

ldnrseek seeks to the relocation entries of the section specified by **sectname.**

ldrseek and **ldnrseek** return SUCCESS or FAILURE. **ldrseek** fails if **sectindx** is greater than the number of sections in the object file; **ldnrseek** fails if there is no section name corresponding with **sectname.** Either function fails if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note that the first section has an index of **one.**

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4)

NAME

ldshread, ldnshread – read an indexednamed section header of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;
```

DESCRIPTION

ldshread reads the section header specified by **sectindx** of the common object file currently associated with **ldptr** into the area of memory beginning at **secthead**.

ldnshread reads the section header specified by **sectname** into the area of memory beginning at **secthead**.

ldshread and **ldnshread** return SUCCESS or FAILURE. **ldshread** fails if **sectindx** is greater than the number of sections in the object file; **ldnshread** fails if there is no section name corresponding with **sectname**. Either function fails if it cannot read the specified section header.

Note that the first section header has an index of **one**.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4)

NAME

ldsseek, ldnsseek – seek to an indexednamed section of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

DESCRIPTION

ldsseek seeks to the section specified by **sectindx** of the common object file currently associated with **ldptr**.

ldnsseek seeks to the section specified by **sectname**.

ldsseek and **ldnsseek** return **SUCCESS** or **FAILURE**. **ldsseek** fails if **sectindx** is greater than the number of sections in the object file; **ldnsseek** fails if there is no section name corresponding with **sectname**. Either function fails if there is no section data for the specified section or if it cannot seek to the specified section.

Note that the first section has an index of **one**.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4)

NAME

ldtbindex – compute the index of a symbol table entry of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex (ldptr)
LDFILE *ldptr;
```

DESCRIPTION

ldtbindex returns the (**long**) index of the symbol table entry at the current position of the common object file associated with **ldptr**.

The index returned by **ldtbindex** may be used in subsequent calls to **ldtbread**(3X). However, since **ldtbindex** returns the index of the symbol table entry that begins at the current position of the object file, if **ldtbindex** is called immediately after a particular symbol table entry has been read, it returns the index of the next entry.

ldtbindex fails if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of 0.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbread(3X), ldtbseek(3X), ldfcn(4)

NAME

ldtbread – read an indexed symbol table entry of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbread (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;
```

DESCRIPTION

ldtbread reads the symbol table entry specified by **symindex** of the common object file currently associated with **ldptr** into the area of memory beginning at **symbol**.

ldtbread returns SUCCESS or FAILURE. **ldtbread** fails if **symindex** is greater than or equal to the number of symbols in the object file, or if it cannot read the specified symbol table entry.

Note that the first symbol in the symbol table has an index of 0.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbseek(3X), ldgetname(3X), ldfcn(4)

NAME

ldtbseek – seek to the symbol table of a common object file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldtbseek (ldptr)
LDFILE *ldptr;
```

DESCRIPTION

ldtbseek seeks to the symbol table of the common object file currently associated with **ldptr**.

ldtbseek returns **SUCCESS** or **FAILURE**. **ldtbseek** fails if the symbol table has been stripped from the object file, or if it cannot seek to the symbol table.

The program must be loaded with the object file access routine library *libld.a*.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbread(3X), ldfcn(4)

NAME

logname – return login name of user

SYNOPSIS

char *logname( )

DESCRIPTION

logname returns a pointer to the null-terminated login name; it extracts the **LOGNAME** environment variable from the user's environment.

This routine is kept in */lib/libPW.a*.

FILES

/etc/profile

SEE ALSO

env(1), login(1), getenv(3C), profile(4), environ(5)

WARNINGS

The return values point to static data whose content is overwritten by each call.

This method of determining a login name is subject to forgery.

NAME

malloc, free, realloc, calloc, mallopt, mallinfo – fast main memory allocator

SYNOPSIS

```
#include <malloc.h>

char *malloc (size)
unsigned size;

void free (ptr)
char *ptr;

char *realloc (ptr, size)
char *ptr;
unsigned size;

char *calloc (nelem, elsize)
unsigned nelem, elsize;

int mallopt (cmd, value)
int cmd, value;

struct mallinfo mallinfo()
```

DESCRIPTION

malloc and **free** provide a simple general-purpose memory allocation package, which runs considerably faster than the **malloc**(3C) package. It is found in the library *malloc*, and is loaded if the option **–lmalloc** is used with **cc**(1) or **ld**(1).

malloc returns a pointer to a block of at least **size** bytes suitably aligned for any use.

The argument to **free** is a pointer to a block previously allocated by **malloc**; after **free** is performed this space is made available for further allocation, and its contents have been destroyed (but see **mallopt** below for a way to change this behavior).

Undefined results will occur if the space assigned by **malloc** is overrun or if some random number is handed to **free**.

realloc changes the size of the block pointed to by **ptr** to **size** bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

calloc allocates space for an array of **nelem** elements of size **elsize**. The space is initialized to zeros.

mallopt provides for control over the allocation algorithm. The available values for **cmd** are:

M_MXFAST Set **maxfast** to **value.** The algorithm allocates all blocks below the size of **maxfast** in large groups and then doles them out very quickly. The default value for **maxfast** is 24.

M_NLBLKS Set **numlblks** to **value.** The above mentioned "large groups" each contain **numlblks** blocks. **numlblks** must be greater than 0. The default value for **numlblks** is 100.

M_GRAIN Set **grain** to **value.** The sizes of all blocks smaller than **maxfast** are considered to be rounded up to the nearest multiple of

grain. **grain** must be greater than 0. The default value of **grain** is the smallest number of bytes which will allow alignment of any data type. Value will be rounded up to a multiple of the default when **grain** is set.

M_KEEP Preserve data in a freed block until the next **malloc, realloc,** or **calloc**. This option is provided only for compatibility with the old version of **malloc** and is not recommended.

These values are defined in the *<malloc.h>* header file.

mallopt may be called repeatedly, but may not be called after the first small block is allocated.

mallinfo provides instrumentation describing space usage. It returns the structure:

```
struct mallinfo  {
        int arena;          /* total space in arena */
        int ordblks;        /* number of ordinary blocks */
        int smblks;         /* number of small blocks */
        int hblkhd;         /* space in holding block headers */
        int hblks;          /* number of holding blocks */
        int usmblks;        /* space in small blocks in use */
        int fsmblks;        /* space in free small blocks */
        int uordblks;       /* space in ordinary blocks in use */
        int fordblks;       /* space in free ordinary blocks */
        int keepcost;       /* space penalty if keep option */
                            /* is used */
}
```

This structure is defined in the *<malloc.h>* header file.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO

brk(2), malloc(3C)

DIAGNOSTICS

malloc, realloc and **calloc** return a **NULL** pointer if there is not enough available memory. When **realloc** returns **NULL**, the block pointed to by **ptr** is left intact. If **mallopt** is called after any allocation or if **cmd** or **value** are invalid, non-zero is returned. Otherwise, it returns zero.

WARNINGS

This package usually uses more data space than **malloc**(3C).

The code size is also bigger than **malloc**(3C).

Note that unlike **malloc**(3C), this package does not preserve the contents of a block when it is freed, unless the M_KEEP option of **mallopt** is used.

Undocumented features of **malloc**(3C) have not been duplicated.

NAME

dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush – database subroutines

SYNOPSIS

```
#include <ndbm.h>

typedef struct {
    char *dptr;
    int dsize;
} datum;

DBM *dbm_open(file, flags, mode)
    char *file;
    int flags, mode;

void dbm_close(db)
    DBM *db;

datum dbm_fetch(db, key)
    DBM *db;
    datum key;

int dbm_store(db, key, content, flags)
    DBM *db;
    datum key, content;
    int flags;

int dbm_delete(db, key)
    DBM *db;
    datum key;

datum dbm_firstkey(db)
    DBM *db;

datum dbm_nextkey(db)
    DBM *db;

int dbm_error(db)
    DBM *db;

int dbm_clearerr(db)
    DBM *db;

void dbm_blkflush(db)
    DBM *db;
```

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions handle very large (a billion blocks) databases and access a keyed item in one or two file system accesses. This package replaces the earlier **dbm**(3x) library, which managed only a single database.

key and **content** are described by the *datum* typedef. A **datum** specifies a string of **dsize** bytes pointed to by **dptr.** Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has *.dir* as its suffix. The second file contains all data and has *.pag* as its suffix.

Before a database can be accessed, it must be opened by **dbm_open.** This opens and/or creates the files *file.dir* and *file.pag* depending on the **flags** parameter (see **open**(2)).

Once open, data stored under a key is accessed by **dbm_fetch** and data is placed under a key by **dbm_store.** The **flags** field can be either **DBM_INSERT** or **DBM_REPLACE. DBM_INSERT** only inserts new entries into the database and will not change an existing entry with the same key. DBM_REPLACE replaces an existing entry if it has the same key. A key (and its associated contents) is deleted by **dbm_delete.** A linear pass through all keys in a database may be made, in an (apparently) random order, by use of **dbm_firstkey** and **dbm_nextkey. dbm_firstkey** returns the first key in the database. **dbm_nextkey** returns the next key in the database. This code traverses the data base:

```
for (key = dbm_firstkey(db); key.dptr != NULL; key =
dbm_nextkey(db))
```

dbm_error returns non-zero on an error in reading or writing the database. **dbm_clearerr** resets the error condition on the named database.

If the same data base is being accessed by multiple programs (using some locking scheme), the routine **dbm_blkflush** can be used to force a new read of internal buffers. This is necessary to insure that no other program has updated the data base and invalidated information stored in internal buffers.

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. A zero indicates no error. Routines that return a **datum** indicate errors with a null (0) **dptr.** If **dbm_store** called with a **flags** value of **DBM_INSERT** finds an existing entry with the same key it returns 1.

NOTES

These are Sequent library routines. Since these are Sequent extensions, code using these routines may not be portable to other systems.

BUGS

The *.pag* file contains holes so that its apparent size is about four times its actual content. Older systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

dptr pointers point into static storage that is changed by subsequent calls. This storage is not necessarily aligned; stored **longs,** for example, should be copied to a properly aligned block of memory before being accessed.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes); and all key/content pairs that hash together must fit on a single block. **dbm_store** returns an error in the event that a disk block fills with inseparable data.

dbm_delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **dbm_firstkey** and **dbm_nextkey** depends on a hashing function, not on anything interesting.

NAME

dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush – database subroutines

SYNOPSIS

```
#include <ndbm.h>

typedef struct {
    char *dptr;
    int dsize;
} datum;

DBM *dbm_open(file, flags, mode)
    char *file;
    int flags, mode;

void dbm_close(db)
    DBM *db;

datum dbm_fetch(db, key)
    DBM *db;
    datum key;

int dbm_store(db, key, content, flags)
    DBM *db;
    datum key, content;
    int flags;

int dbm_delete(db, key)
    DBM *db;
    datum key;

datum dbm_firstkey(db)
    DBM *db;

datum dbm_nextkey(db)
    DBM *db;

int dbm_error(db)
    DBM *db;

int dbm_clearerr(db)
    DBM *db;

void dbm_blkflush(db)
    DBM *db;
```

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions handle very large (a billion blocks) databases and access a keyed item in one or two file system accesses. This package replaces the earlier **dbm**(3x) library, which managed only a single database.

key and **content** are described by the *datum* typedef. A **datum** specifies a string of **dsize** bytes pointed to by **dptr.** Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has *.dir* as its suffix. The second file contains all data and has *.pag* as its suffix.

Before a database can be accessed, it must be opened by **dbm_open**. This opens and/or creates the files *file.dir* and *file.pag* depending on the **flags** parameter (see **open**(2)).

Once open, data stored under a key is accessed by **dbm_fetch** and data is placed under a key by **dbm_store**. The **flags** field can be either **DBM_INSERT** or **DBM_REPLACE. DBM_INSERT** only inserts new entries into the database and will not change an existing entry with the same key. DBM_REPLACE replaces an existing entry if it has the same key. A key (and its associated contents) is deleted by **dbm_delete**. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of **dbm_firstkey** and **dbm_nextkey**. **dbm_firstkey** returns the first key in the database. **dbm_nextkey** returns the next key in the database. This code traverses the data base:

```
for (key = dbm_firstkey(db); key.dptr != NULL; key =
dbm_nextkey(db))
```

dbm_error returns non-zero on an error in reading or writing the database. **dbm_clearerr** resets the error condition on the named database.

If the same data base is being accessed by multiple programs (using some locking scheme), the routine **dbm_blkflush** can be used to force a new read of internal buffers. This is necessary to insure that no other program has updated the data base and invalidated information stored in internal buffers.

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. A zero indicates no error. Routines that return a **datum** indicate errors with a null (0) **dptr.** If **dbm_store** called with a **flags** value of **DBM_INSERT** finds an existing entry with the same key it returns 1.

NOTES

These are Sequent library routines. Since these are Sequent extensions, code using these routines may not be portable to other systems.

BUGS

The *.pag* file contains holes so that its apparent size is about four times its actual content. Older systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

dptr pointers point into static storage that is changed by subsequent calls. This storage is not necessarily aligned; stored **longs,** for example, should be copied to a properly aligned block of memory before being accessed.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes); and all key/content pairs that hash together must fit on a single block. **dbm_store** returns an error in the event that a disk block fills with inseparable data.

dbm_delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **dbm_firstkey** and **dbm_nextkey** depends on a hashing function, not on anything interesting.

NAME

plot – graphics interface subroutines

SYNOPSIS

```
openpl ()

erase ()

label (s)
char *s;

line (x1, y1, x2, y2)
int x1, y1, x2, y2;

circle (x, y, r)
int x, y, r;

arc (x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;

move (x, y)
int x, y;

cont (x, y)
int x, y;

point (x, y)
int x, y;

linemod (s)
char *s;

space (x0, y0, x1, y1)
int x0, y0, x1, y1;

closepl ()
```

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. **space** must be used before any of these functions to declare the amount of space necessary (see **plot**(4)). **openpl** must be used before any of the others to open the device for writing. **closepl** flushes the output.

circle draws a circle of radius **r** with center at the point **(x, y)**.

arc draws an arc of a circle with center at the point **(x, y)** between the points **(x0, y0)** and **(x1, y1)**.

String arguments to **label** and **linemod** are terminated by nulls and do not contain new-lines.

See **plot**(4) for a description of the effect of the remaining functions.

The library files listed below provide several flavors of these routines.

FILES

/usr/lib/libplot.a	produces output for **tplot**(1G) filters
/usr/lib/lib300.a	for DASI 300
/usr/lib/lib300s.a	for DASI 300s
/usr/lib/lib450.a	for DASI 450

/usr/lib/lib4014.a for TEKTRONIX 4014

SEE ALSO

graph(1G), stat(1G), tplot(1G), plot(4)

WARNINGS

In order to compile a program containing these functions in *file.c* it is necessary to use `cc file.c -lplot`.

In order to execute it, it is necessary to use `a.out | tplot`.

The above routines use *<stdio.h>*, which causes them to increase the size of programs, not otherwise using standard I/O more than might be expected.

NAME

putspent – write shadow password file entry

SYNOPSIS

```
#include <shadow.h>

int putspent (p, fp)
struct spwd *p;
FILE *fp;
```

DESCRIPTION

The **putspent** routine is the inverse of **getspent**(3X). Given a pointer to a *spwd* structure created by the **getspent** routine (or the **getspnam** routine), the **putspent** routine writes a line on the stream *fp*, which matches the format of */etc/shadow*.

If the **sp_min**, **sp_max**, or **sp_lstchg** field of the *spwd* structure is –1, the corresponding */etc/shadow* field is cleared.

SEE ALSO

getspent(3X).

DIAGNOSTICS

The *putspent* routine returns non-zero if an error was detected during its operation, otherwise zero.

WARNINGS

If a program not otherwise using standard I/O uses this routine, the size of the program will increase more than might be expected.

This routine is for internal use only, compatibility is not guaranteed.

NAME

regcmp, regex – compile and execute regular expression

SYNOPSIS

char *regcmp (string1 [, string2, ...], (char *)0)
char *string1, *string2, ...;

char *regex (re, subject[, ret0, ...])
char *re, *subject, *ret0, ...;

extern char *__loc1;

DESCRIPTION

regcmp compiles a regular expression (consisting of the concatenated arguments) and returns a pointer to the compiled form. **malloc**(3C) is used to create space for the compiled form. It is the user's responsibility to free unneeded space so allocated. A **NULL** return from **regcmp** indicates an incorrect argument. **regcmp**(1) has been written to generally preclude the need for this routine at execution time.

regex executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. **regex** returns **NULL** on failure or a pointer to the next unmatched character on success. A global character pointer **_loc1** points to where the match began. **regcmp** and **regex** were mostly borrowed from the editor, **ed**(1); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

[ ] * . ^ These symbols retain their meaning in **ed**(1).

$ Matches the end of the string; **\n** matches a new-line.

– Within brackets the minus means *through*. For example, **[a–z]** is equivalent to [abcd...xyz]. The – can appear as itself only if used as the first or last character. For example, the character class expression []–] matches the characters] and –.

+ A regular expression followed by + means *one or more times*. For example, [0–9]+ is equivalent to [0–9] [0–9]*.

{m} {m,} {m,u}
Integer values enclosed in {} indicate the number of times the preceding regular expression is to be applied. The value **m** is the minimum number and **u** is a number, less than 256, which is the maximum. If only **m** is present (e.g., **{m}**), it indicates the exact number of times the regular expression is to be applied. The value **{m,}** is analogous to **{m,infinity}**. The plus (+) and star (*) operations are equivalent to {1,} and {0,} respectively.

(...)$ The value of the enclosed regular expression is to be returned. The value will be stored in the **(n+1)**th argument following the subject argument. At most ten enclosed regular expressions are allowed. **regex** makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, e.g., *, +, {}, can work on a single character or a regular expression enclosed in parentheses. For example, `(a*(cb+)*)$0`.

By necessity, all the above defined symbols are special. They must, therefore, be escaped with a \ (backslash) to be used as themselves.

EXAMPLES

Example 1:

```
char *cursor, *newcursor, *ptr;
     ...
newcursor = regex((ptr = regcmp("^\n", (char *)0)), cursor);
free(ptr);
```

This example will match a leading new-line in the subject string pointed at by **cursor**.

Example 2:

```
char ret0[9];
char *newcursor, *name;
     ...
name = regcmp("([A-Za-z][A-za-z0-9]{0,7})$0", (char *)0);
newcursor = regex(name, "012Testing345", ret0);
```

This example will match through the string **"Testing3"** and will return the address of the character after the last matched character (the 4). The string **"Testing3"** will be copied to the character array **ret0**.

Example 3:

```
#include "file.i"
char *string, *newcursor;
     ...
newcursor = regex(name, string);
```

This example applies a precompiled regular expression in **file.i** [see **regcmp**(1)] against **string**.

These routines are kept in */lib/libPW.a*.

SEE ALSO

ed(1), regcmp(1), malloc(3C), regex(3x)

BUGS

The user program may run out of memory if **regcmp** is called iteratively without freeing the vectors no longer required.

NAME

sputl, sgetl – access long integer data in a machine-independent fashion

SYNOPSIS

```
void sputl (value, buffer)
long value;
char *buffer;

long sgetl (buffer)
char *buffer;
```

DESCRIPTION

sputl takes the four bytes of the long integer **value** and places them in memory starting at the address pointed to by **buffer**. The ordering of the bytes is the same across all machines.

sgetl retrieves the four bytes in memory starting at the address pointed to by **buffer** and returns the long integer value in the byte ordering of the host machine.

The combination of **sputl** and **sgetl** provides a machine-independent way of storing long numeric data in a file in binary form without conversion to characters.

A program that uses these functions must be loaded with the object-file access routine library *libld.a*.

NAME

ut_add_user, ut_delete_user – add/delete users in utmp file

SYNOPSIS

```
#include <utmp.h>

struct utmp *ut_add_user (user, line, pid, host)
char *user;
char *line;
int pid;
char *host;

struct utmp *ut_delete_user (line, pid, e_term, e_status)
char *line;
int pid;
int e_term;
int e_status;
```

DESCRIPTION

Programs that need to add or delete entries from the */etc/utmp* file should go through these interfaces.

Users may be added due to connections from remote machines (for example, rlogin), or windowing systems (for example, the X Window System).

user is a user login name up to 8 characters.

line is the user login line returned from the connection service. The leading 4 characters of */dev/* are not included, but the remainder of the path is used. This string must be no longer than 12 characters.

pid is the process ID as returned from the **getpid** system call.

host describes the origin of the login session; the four cases are as follows:

A login from a hard-wired line must use a **NULL** character ((**char** *)0) as the **host** parameter.

A remote login (through a network) must use the name of the host being logged in from as the **host** parameter.

A windowing system using pseudoterminals via a network must use the window system display name as the **host** parameter.

A windowing system using pseudoterminals via a hard-wired line must use the hard-wired line as the **host** parameter.

The routine **ut_add_user** creates the *utmp* entry with type USER_PROCESS (see *utmp.h*). The routine **ut_delete_user** is used when a login session is completed. It changes the **utmp** entry type to DEAD_PROCESS. **ut_delete_user** finds an entry based on line name or process ID. If the line name is not **NULL**, it is used to search the *utmp* file and locate the entry. If the line name is **NULL**, the process ID supplied is used to do the lookup. The parameter **e_termination** should contain the lower 8 bits of the process exit status, and **e_exit** should contain the upper 8 bits of the process exit status.

FILES

/etc/utmp
/etc/wtmp

/etc/utmp_dbm.dir
/etc/utmp_dbm.pag

SEE ALSO

getut(3C)

DIAGNOSTICS

On success, these routines return a pointer to the **utmp** structure placed in the *utmp* file. The pointer is normally be written to the **wtmp** file by the program invoking these routines.

A **NULL** pointer is returned upon failure to read or write, regardless of whether the failure was because of permission or from another system failure.

The routine **ut_add_user** may also return a **NULL** pointer in the event that the system login limit is reached. In this case a message stating that the login limit has been reached is printed on *stdout*. No entry is placed in */etc/utmp* and the calling program should exit without establishing a login session.

NOTES

For enhanced performance reasons, Sequent uses a hashed *utmp* file. These routines are not part of standard System V and code that uses them may not be portable to other systems.